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

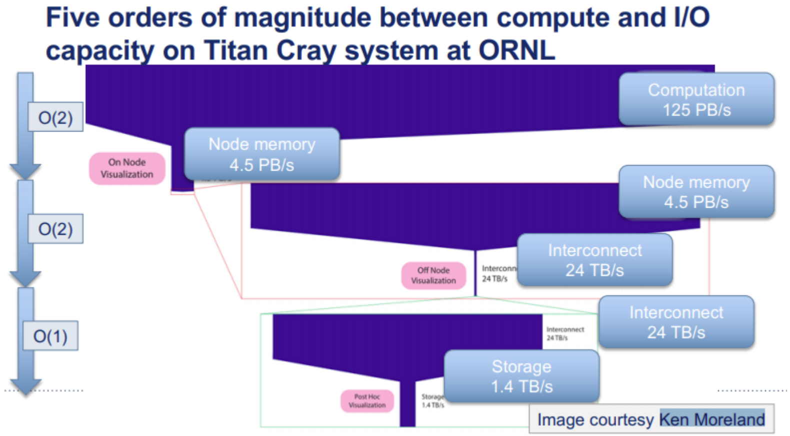
esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

Outline

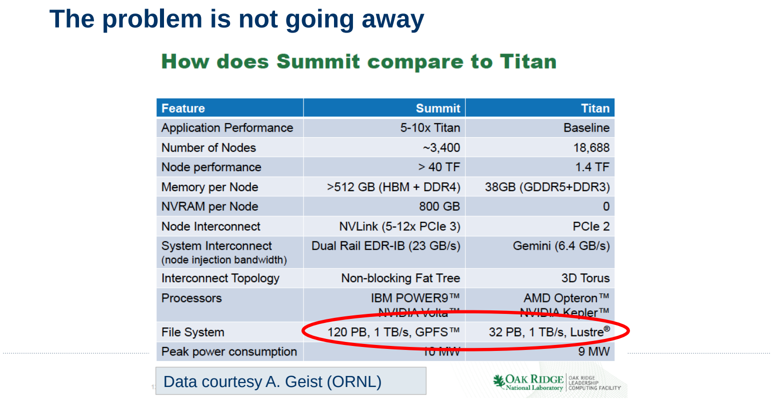
Learning Objectives

- Describe the role of middleware and file formats (Section Middleware)
- Discuss challenges for data-driven research (Data)
- Identify typical I/O performance issues and their causes (Section Performance)
- Apply performance models to assess and optimize the application I/O performance (Section Model)
- Design a data model for NetCDF/CF (Section NetCDF)
- Execute programs in C and Python that read and write NetCDF files in a metadata-aware manner (Section Progs)
- Analyze, manipulate and visualise NetCDF data (Section NetCDF2)
- Implement an application that utilizes parallel I/O to store and analyze data (Section Parallel I/O)
- Describe ongoing research activities in high-performance storage (Section Storage)

I/O Bottleneck



I/O Bottleneck



The problem is not going away

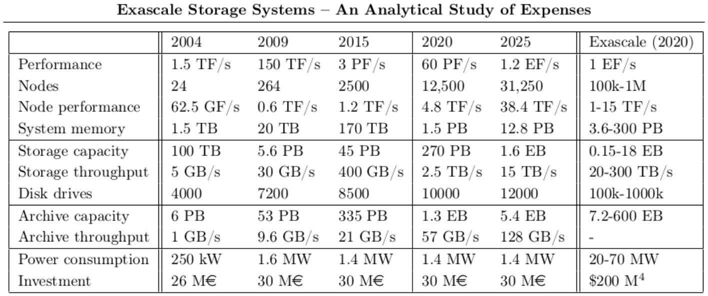
How does Summit compare to Titan

Feature	Summit	Titan
Application Performance	5-10x Titan	Baseline
Number of Nodes	~3,400	18,688
Node performance	> 40 TF	1.4 TF
Memory per Node	>512 GB (HBM + DDR4)	38GB (GDDR5+DDR3)
NVRAM per Node	800 GB	0
Node Interconnect	NVLink (5-12x PCIe 3)	PCIe 2
System Interconnect (node injection bandwidth)	Dual Rail EDR-IB (23 GB/s)	Gemini (6.4 GB/s)
Interconnect Topology	Non-blocking Fat Tree	3D Torus
Processors	IBM POWER9™ NVIDIA Volta™	AMD Opteron™ NVIDIA Kepler™
File System	120 PB, 1 TB/s, GPFS™	32 PB, 1 TB/s, Lustre®
Peak power consumption	10 MW	9 MW

Data courtesy A. Geist (ORNL)

I/O Bottleneck

Kunkel et al. [105] 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.

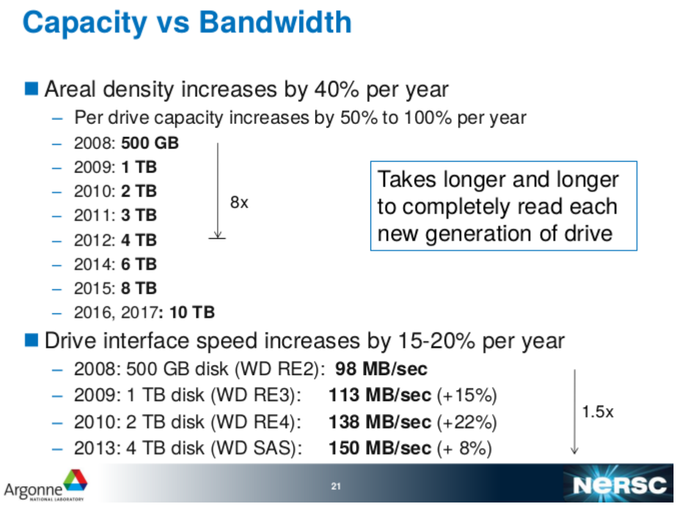


Exascale Storage Systems – An Analytical Study of Expenses

	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

I/O Bottleneck



Capacity vs Bandwidth

- Areal density increases by 40% per year
  - Per drive capacity increases by 50% to 100% per year
  - 2008: **500 GB**
  - 2009: **1 TB**
  - 2010: **2 TB**
  - 2011: **3 TB**
  - 2012: **4 TB**
  - 2014: **6 TB**
  - 2015: **8 TB**
  - 2016, 2017**: 10 TB**

$\downarrow$ 8x

Takes longer and longer to completely read each new generation of drive

- Drive interface speed increases by 15-20% per year
  - 2008: 500 GB disk (WD RE2): **98 MB/sec**
  - 2009: 1 TB disk (WD RE3): **113 MB/sec** (+15%)
  - 2010: 2 TB disk (WD RE4): **138 MB/sec** (+22%)
  - 2013: 4 TB disk (WD SAS): **150 MB/sec** (+ 8%)

$\downarrow$ 1.5x

Argonne NATIONAL LABORATORY 21 NERSC

Input/Output

Input/Output (I/O) is simply data migration.

- Memory ⇔ Disk

I/O is a very expensive operation.

- Interactions with data in memory and on disk.

How is I/O performed?

- I/O Pattern
  - Number of processes and files.
  - Characteristics of file access.

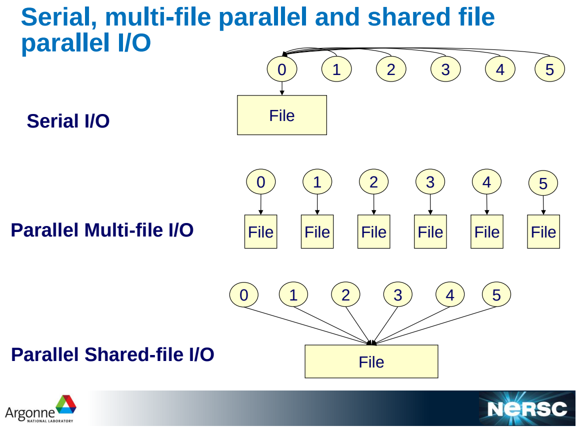
Where is I/O performed?

- Characteristics of the computational system.
- Characteristics of the file system.

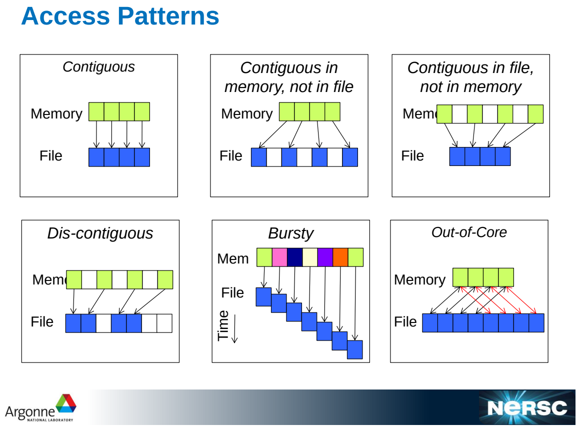
I/O Performance

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

I/O Types


Serial, multi-file parallel and shared file parallel I/O

I/O Patterns


Access Patterns
Contiguous
Memory
File
Contiguous in memory, not in file
Memory
File
Contiguous in file, not in memory
Mem
File
Dis-contiguous
Mem
File
Bursty
Mem
File
Time
Out-of-Core
Memory
File
Argonne NATIONAL LABORATORY
NERSC

I/O Probs

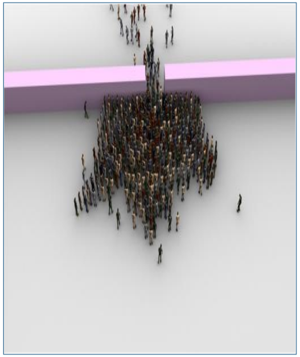
What are the problems?

Not enough I/O capacity on current HPC systems, and the trend is getting worse.

If there's not enough I/O, you can't write data to storage, so you can't 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.



Challenges in Application I/O

Challenges in Application I/O

- Leveraging aggregate communication and I/O bandwidth of clients
  - …but not overwhelming a resource limited I/O system with uncoordinated accesses!
- Limiting number of files that must be managed
  - Also a performance issue
- Avoiding unnecessary post-processing
- Often application teams spend so much time on this that they never get any further:
  - Interacting with storage through convenient abstractions
  - Storing in portable formats

Data Storage

- 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.
- 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.
- 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.

Middleware Level

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

Application Level

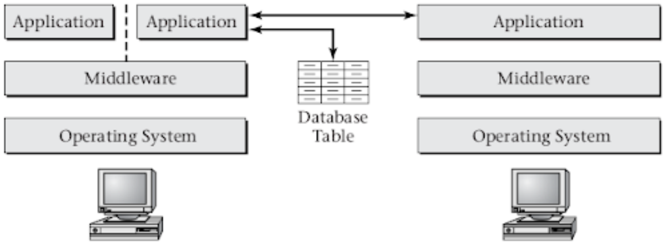
- In-situ analysis
  - In biology and biomedical engineering, in situ means to examine the phenomenon exactly in place where it occurs (i.e., without moving it to some special medium).
  - Rather than applications writing their raw output to storage to later be read again for post-processing (e.g., visualization, filtering, statistics), in-situ processing removes this overhead by performing the analysis directly on the same machines as where the applications run.
  - ParaView [67], Dax [130], and Damaris/Viz [59] are tools for large-scale in-situ visualization.

Discussion

- 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 such as ADIOS.
- In-situ analysis is an example of how application-specific solutions can be used to improve I/O throughput and thus application performance.
- No one-size-fits-all solution to the storage problem and programmers must take I/O into careful consideration when developing applications.

Middleware

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



- Common middleware examples include database middleware, application server middleware, message-oriented middleware, web middleware, and transaction-processing monitors.

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, naming, and communication.
- Relational database systems is one example of middleware.
  - Such systems provide a more sophisticated form of persistent storage than the files supported by most operating systems.

Describe the role of middleware and file formats

- File formats
  -
  -
-
  -

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.

Challenges for Data-driven Research

- 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.
- Many processes still require experts. For example, porting a workflow from one system to another still requires adjusting runtime parameters of applications and deciding on how data is managed.
- Data-driven workflows may benefit from the explicit and simultaneous use of a locally heterogeneous set of computing and storage technologies.

I/O Path

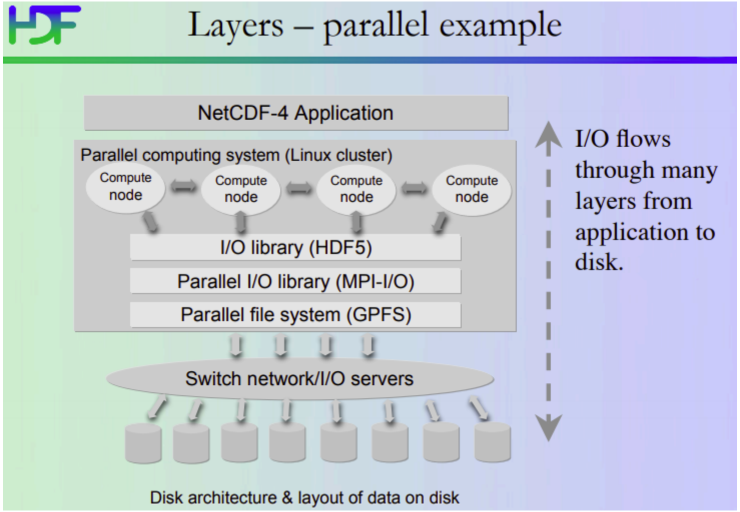
The I/O path describes all the layers (and components) involved in I/O and how they interact. In brief, the I/O path of a write operation can be described as follows:

- When an application issues an I/O call, data is copied between the user-space buffer and the page cache that is offered in kernel-space.
- In kernel-space memory the data is cached and write operations can be deferred as long as free memory is available.
- At this point the write call can complete, because a programmer cannot modify kernel-space directly.
- A scheduler inside the kernel decides when modified pages are transferred to the block device.
- The mapping from offsets in logical files to addresses on the block storage is managed by a file system. Further, file systems provide the hierarchical namespace and offer additional management features.

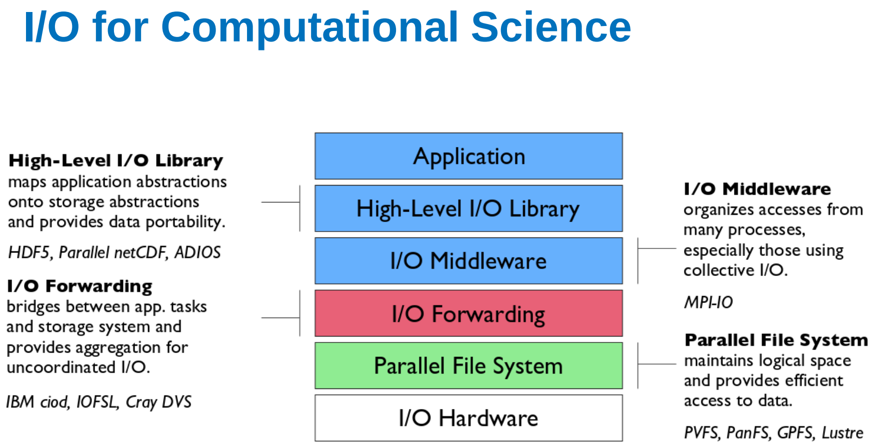
I/O Performance

- 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 modelling and simulation techniques.

I/O Layers


Layers – parallel example
NetCDF-4 Application
Parallel computing system (Linux cluster)
Compute node
Compute node
Compute node
Compute node
I/O library (HDF5)
Parallel I/O library (MPI-I/O)
Parallel file system (GPFS)
Switch network/I/O servers
Disk architecture & layout of data on disk
I/O flows through many layers from application to disk.

I/O Stack



I/O for Computational Science

High-Level I/O Library
maps application abstractions onto storage abstractions and provides data portability.

HDF5, Parallel netCDF, ADIOS

I/O Forwarding
bridges between app. tasks and storage system and provides aggregation for uncoordinated I/O.

IBM ciod, IOFSL, Cray DVS

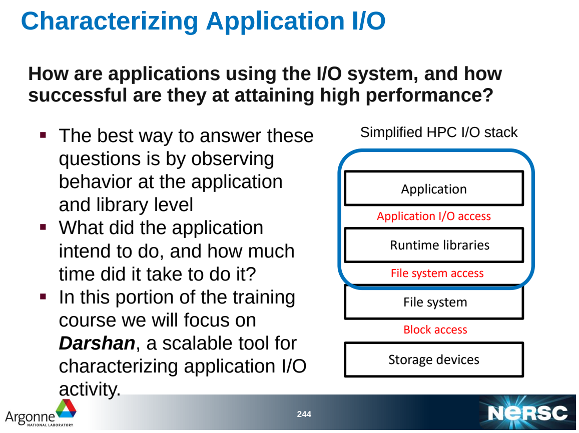
I/O Middleware
organizes accesses from many processes, especially those using collective I/O.

MPI-IO

Parallel File System
maintains logical space and provides efficient access to data.

PVFS, PanFS, GPFS, Lustre

I/O Stack



Characterizing Application I/O

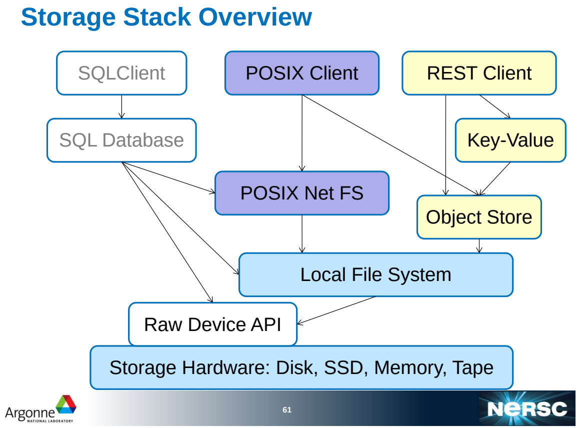
How are applications using the I/O system, and how successful are they at attaining high performance?

- The best way to answer these questions is by observing behavior at the application and library level
- What did the application intend to do, and how much time did it take to do it?
- In this portion of the training course we will focus on ***Darshan***, a scalable tool for characterizing application I/O activity.

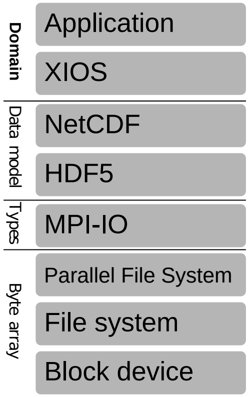
Simplified HPC I/O stack

244

Storage Stack



Storage Stack



Typical I/O Performance Issues

Access patterns:

- Access granularity
- Randomness
- Concurrency
- Load balance
- Access type
- Predictability

Typical I/O Performance Issues

I/O strategy:

- Caching algorithm
- Replication
- High availability (HA) support
- I/O forwarding
- Aggregation
- Scheduling

Typical I/O Performance Issues

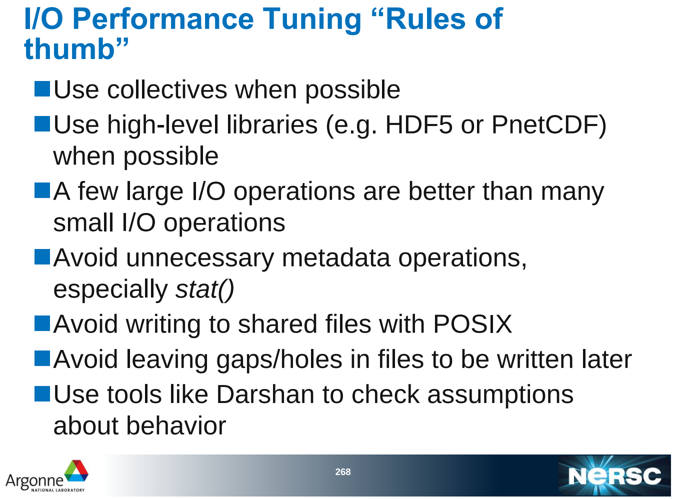
Parallel file system:

- Design
- Implementation
- Resource consumption
- Communication
- Data distribution
- Metadata handling
- Consistency semantics
- Locking strategy

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 subsystems
  - 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?

I/O Performance Tuning "Rules of Thumb"

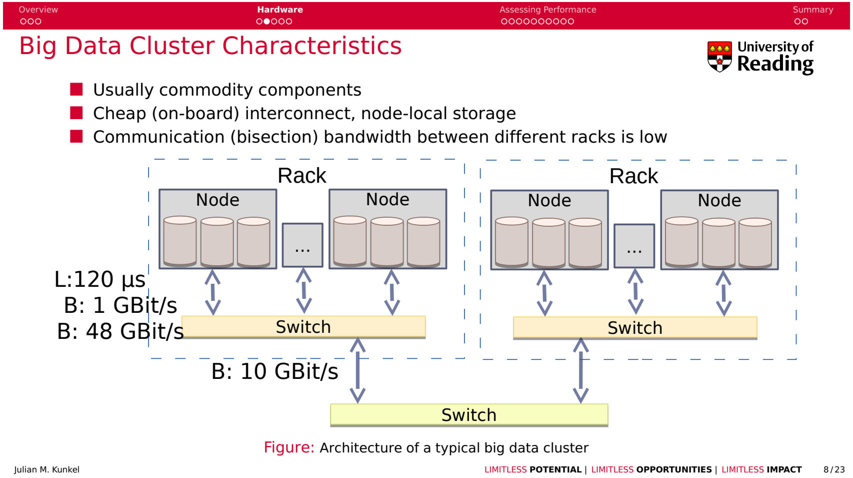


I/O Performance Tuning "Rules of thumb"

- 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

Argonne NATIONAL LABORATORY 268 NERSC

I/O Performance Model



Big Data Cluster Characteristics

- Usually commodity components
- Cheap (on-board) interconnect, node-local storage
- Communication (bisection) bandwidth between different racks is low

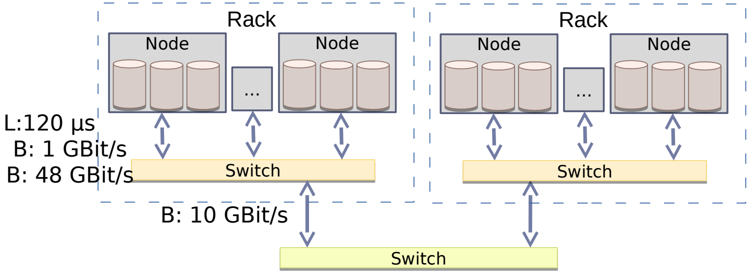
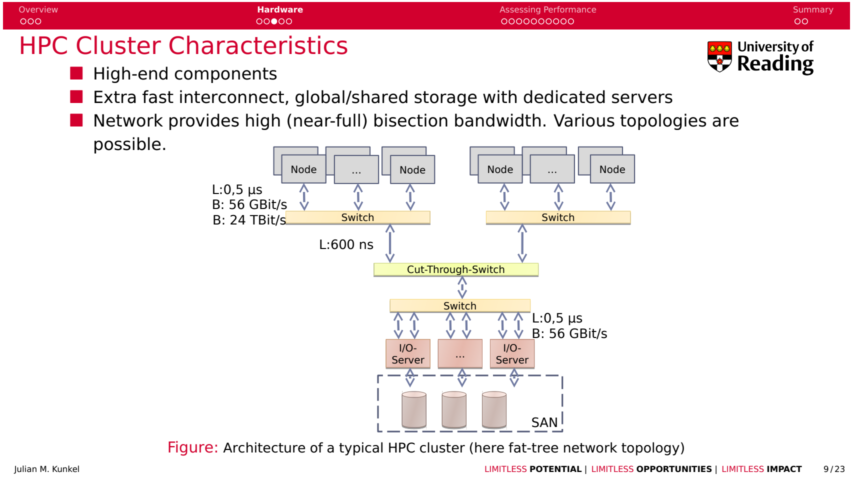


Figure: Architecture of a typical big data cluster

I/O Performance Model



HPC Cluster Characteristics

- High-end components
- Extra fast interconnect, global/shared storage with dedicated servers
- Network provides high (near-full) bisection bandwidth. Various topologies are possible.

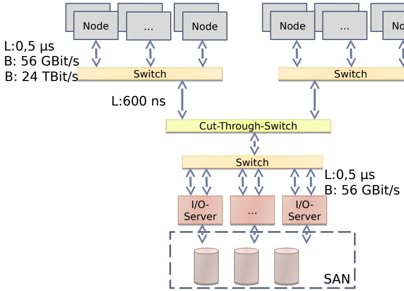
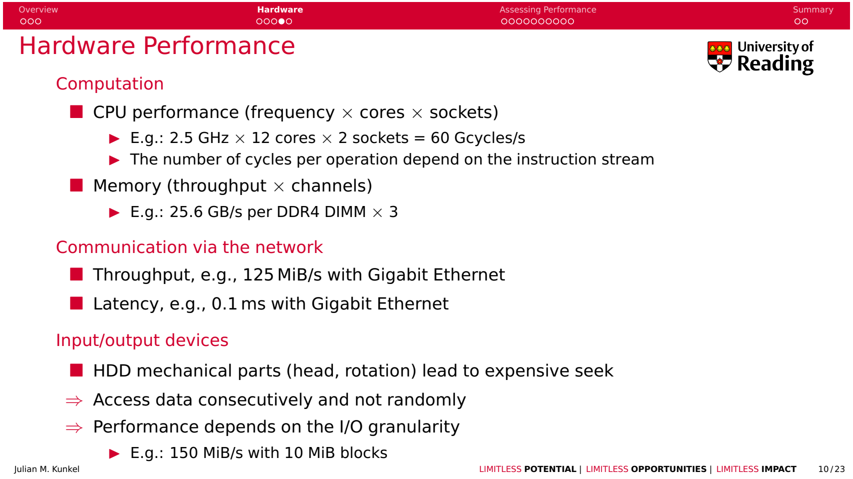


Figure: Architecture of a typical HPC cluster (here fat-tree network topology)

I/O Performance Model



Hardware Performance

Computation

- CPU performance (frequency × cores × sockets)
  - E.g.: 2.5 GHz × 12 cores × 2 sockets = 60 Gcycles/s
  - The number of cycles per operation depend on the instruction stream
- Memory (throughput × channels)
  - E.g.: 25.6 GB/s per DDR4 DIMM × 3

Communication via the network

- Throughput, e.g., 125 MiB/s with Gigabit Ethernet
- Latency, e.g., 0.1 ms with Gigabit Ethernet

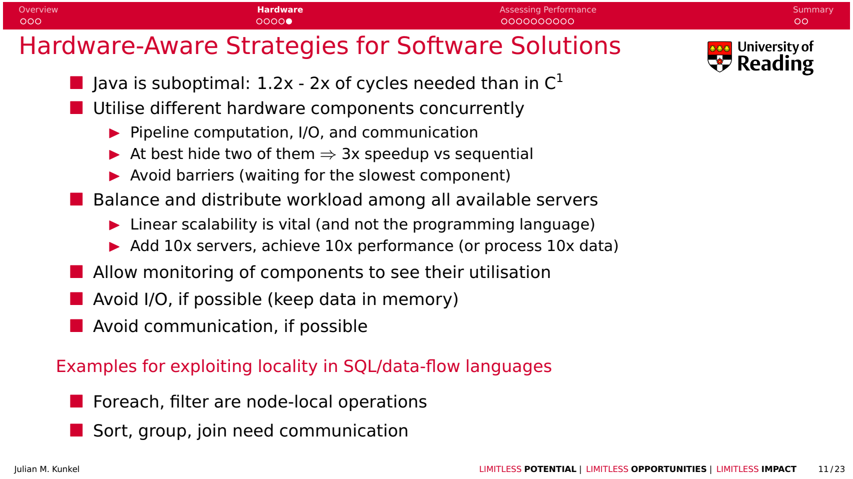
Input/output devices

- HDD mechanical parts (head, rotation) lead to expensive seek

⇒ Access data consecutively and not randomly

⇒ Performance depends on the I/O granularity
  - E.g.: 150 MiB/s with 10 MiB blocks

I/O Performance Model



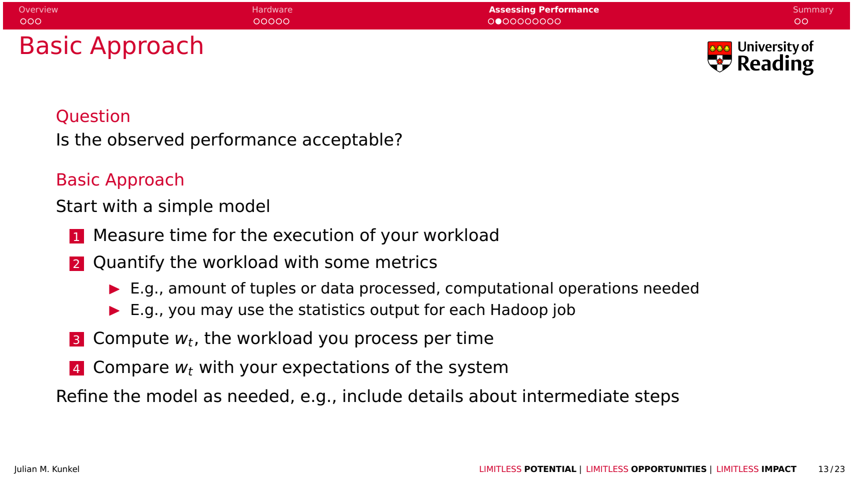
Hardware-Aware Strategies for Software Solutions

- Java is suboptimal: 1.2x - 2x of cycles needed than in C[1]
- Utilise different hardware components concurrently
  - Pipeline computation, I/O, and communication
  - At best hide two of them ⇒ 3x speedup vs sequential
  - Avoid barriers (waiting for the slowest component)
- Balance and distribute workload among all available servers
  - Linear scalability is vital (and not the programming language)
  - Add 10x servers, achieve 10x performance (or process 10x data)
- Allow monitoring of components to see their utilisation
- Avoid I/O, if possible (keep data in memory)
- Avoid communication, if possible

Examples for exploiting locality in SQL/data-flow languages

- Foreach, filter are node-local operations
- Sort, group, join need communication

I/O Performance Model



Basic Approach

Question

Is the observed performance acceptable?

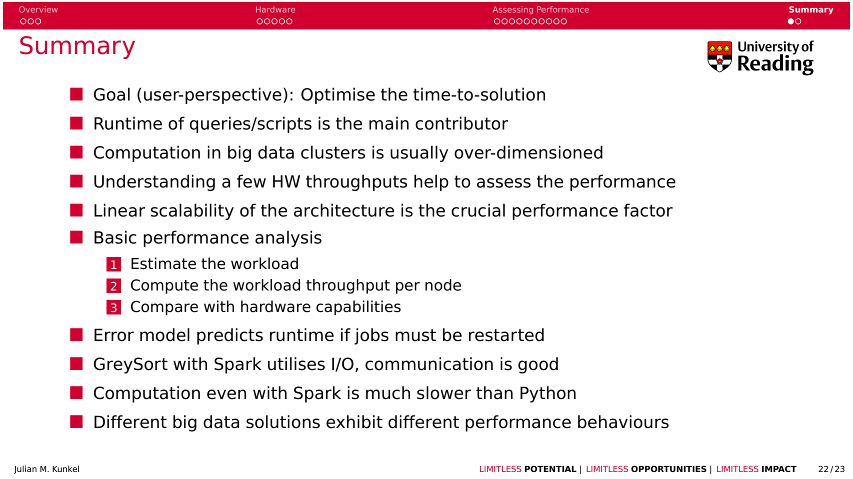
Basic Approach

Start with a simple model

1. Measure time for the execution of your workload
2. Quantify the workload with some metrics
   - E.g., amount of tuples or data processed, computational operations needed
   - E.g., you may use the statistics output for each Hadoop job
3. Compute w_t, the workload you process per time
4. Compare w_t with your expectations of the system

Refine the model as needed, e.g., include details about intermediate steps

I/O Performance Model



Summary

- Goal (user-perspective): Optimise the time-to-solution
- Runtime of queries/scripts is the main contributor
- Computation in big data clusters is usually over-dimensioned
- Understanding a few HW throughputs help to assess the performance
- Linear scalability of the architecture is the crucial performance factor
- Basic performance analysis
  1. Estimate the workload
  2. Compute the workload throughput per node
  3. Compare with hardware capabilities
- Error model predicts runtime if jobs must be restarted
- GreySort with Spark utilises I/O, communication is good
- Computation even with Spark is much slower than Python
- Different big data solutions exhibit different performance behaviours

NetCDF

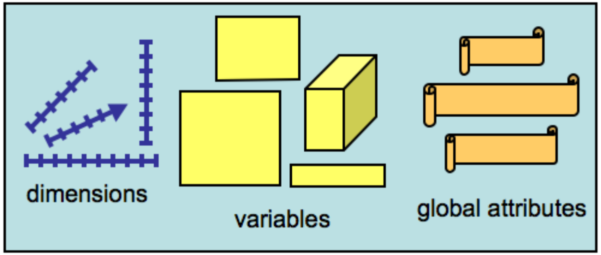
- In a simple view, NetCDF is:
  - A data mode.
  - A file format.
  - A set of APIs and libraries for various programming languages.
- Together, the data model, file format, and APIs support the creation, access, and sharing of scientific data.
- NetCDF allows the user to describe multidimensional data 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.
- The NetCDF system includes utilities for producing human-oriented CDL text files from binary NetCDF datasets and vice-versa.
  - Example.

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.



The Classic NetCDF Model Characteristics

- A NetCDF file has named variables, attributes, and dimensions
- Variables are for data, attributes are for metadata (data about data)
- Dimensions are for specifying shapes of variables
- Attributes may apply to a whole file or to a single variable
- Variables may share dimensions, indicating a common grid
- One dimension may be of unlimited length
- Each variable or attribute has one of six types: char, byte, short, int, float, double

The Classic NetCDF Model Limitations

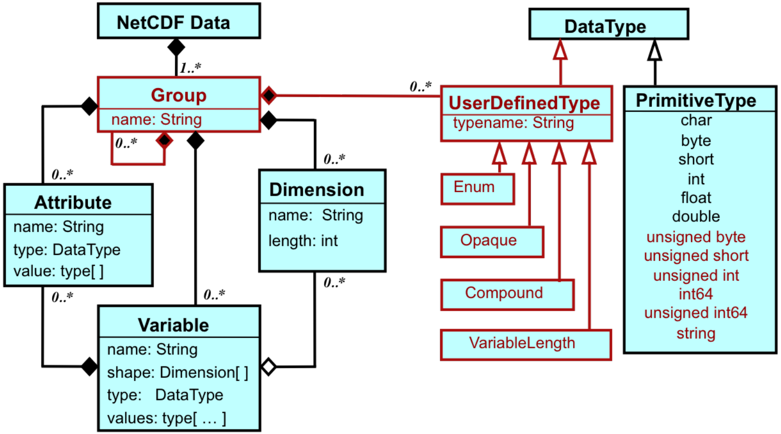
- No real data structures, just multidimensional arrays and lists
- No nested structures, variable-length types, or ragged arrays
- Only one shared unlimited dimension for appending new data
- A flat name space for dimensions and variables
- Character arrays rather than "real" strings
- A small set of numeric types

The Classic NetCDF Format Limitations

- Large variables must be less than 4 GB (per record)
- No real compression supported, just scale/offset packing
- Changing a file schema (the logical structure of the file) may be very inefficient
- I/O is serial in Unidata NetCDF-3 (but see Argonne/Northwestern Parallel NetCDF project)
- Efficient access requires data for a variable to be written and read contiguously

NetCDF Data Models

- Classic: Simplest model – dimensions, variables, attributes
- Enhanced: More powerful model – adds groups, types, nesting



The NetCDF-4 Enhanced Data Model

- The new data model, which is known as the "Common Data Model" is part of an effort here at Unidata to find a common engineering language for the development of scientific data solutions.
- It 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

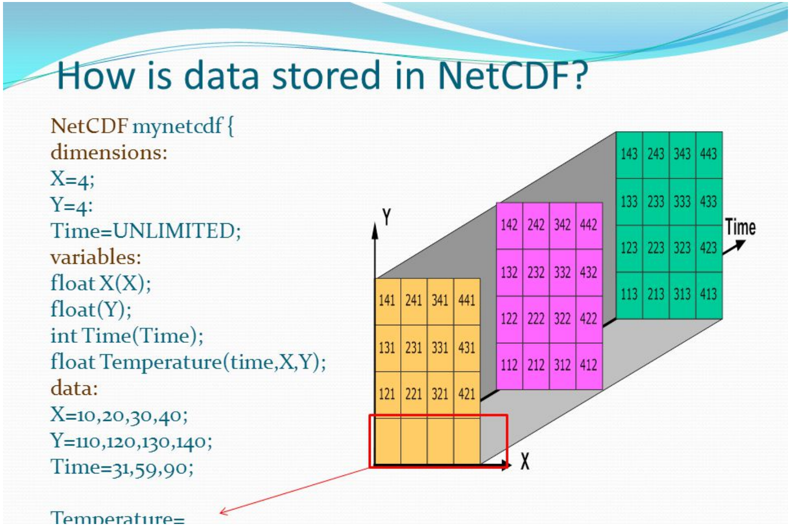
- A file has a top-level unnamed group.
- Each group may contain one or more named subgroups, user-defined types, variables, dimensions, and attributes.
- Variables also have attributes.
- Variables may share dimensions, indicating a common grid.
- One or more dimensions may be of unlimited length.

NetCDF-4 and HDF5

NetCDF-4 uses HDF5 as a storage layer:

- Provide performance advantages of HDF5
- Compression
- Chunking
- Efficient schema changes
- Useful for very large or complex data
- Suitable for high-performance computing

NetCDF Data Model



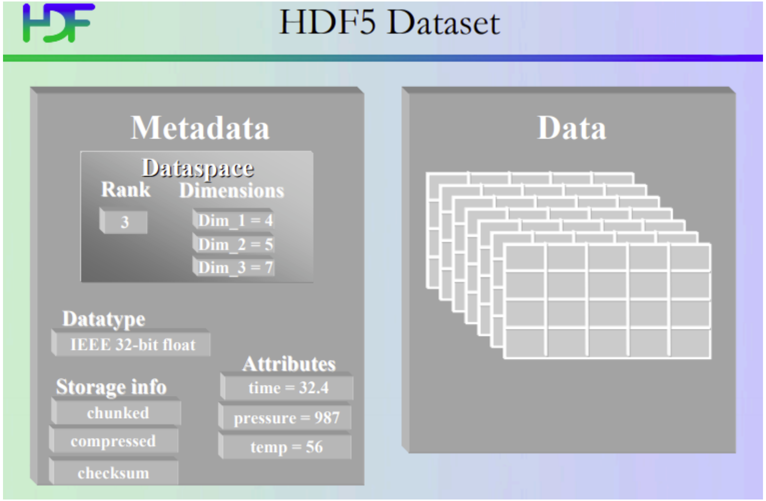
How is data stored in NetCDF?

```
NetCDF mynetcdf {
dimensions:
X=4;
Y=4:
Time=UNLIMITED;
variables:
float X(X);
float(Y);
int Time(Time);
float Temperature(time,X,Y);
data:
X=10,20,30,40;
Y=110,120,130,140;
Time=31,59,90;

Temperature=
```

Y
X
Time
141 241 341 441
131 231 331 431
121 221 321 421
142 242 342 442
132 232 332 432
122 222 322 422
112 212 312 412
143 243 343 443
133 233 333 433
123 223 323 423
113 213 313 413

HDF5 Data Model



ESDM Data Model

Experience-based "Best Practices" for Writing NetCDF Files

- Conventions
  - Developers should be familiar with and use existing NetCDF conventions, when practical.
- 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 CF conventions define metadata that are included in the same file as the data (thus making the file “self-describing”).
- The purpose of the CF conventions is to require conforming datasets to contain sufficient metadata that they are self-describing in the sense that each variable in the file has an associated description of what it represents, including physical units if appropriate, and that 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.

Execute programs in C and Python that read and write NetCDF files in a metadata-aware manner

- Lab session, different presentation
- Luciana (C) + Sadie (Python)
-

Analyze, manipulate and visualise NetCDF data

- Lab session, different presentation
- Luciana (C) + Sadie (Python)

Implement an application that utilizes parallel I/O to store and analyze data

- Lab session, different presentation
- Luciana (C) + Sadie (Python)

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

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**