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

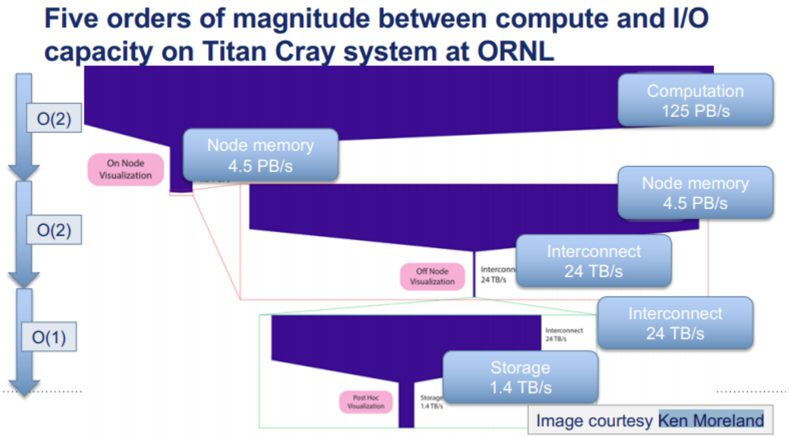
Outline

Learning Objectives

- 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

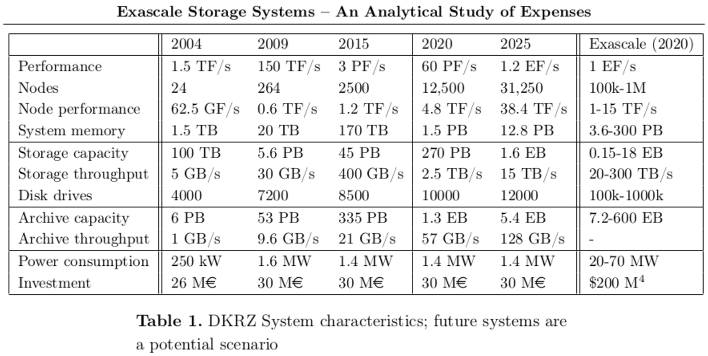
I/O Bottleneck – Example



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.

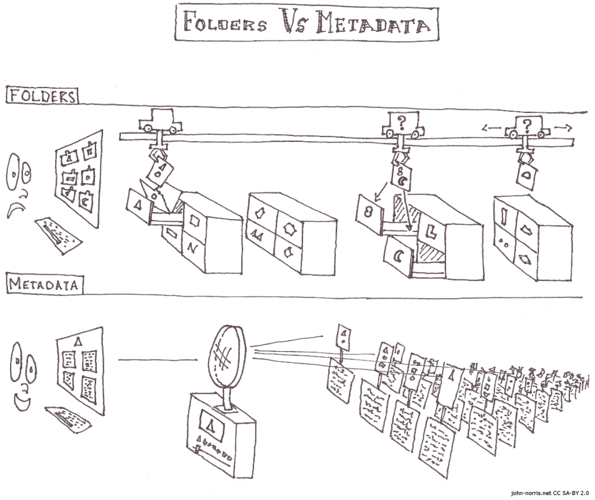


Exascale Storage Systems – An Analytical Study of Expenses

	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 Meta



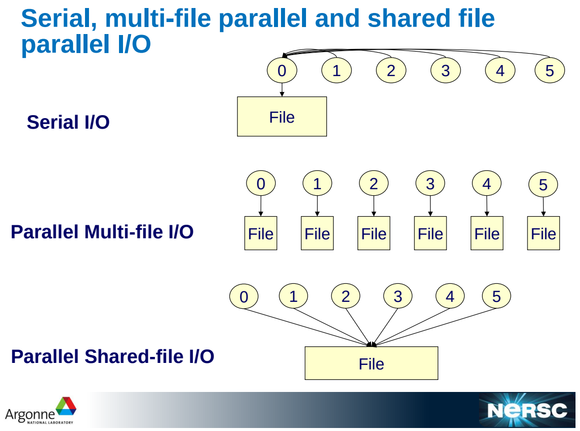
Input/Output

- 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


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

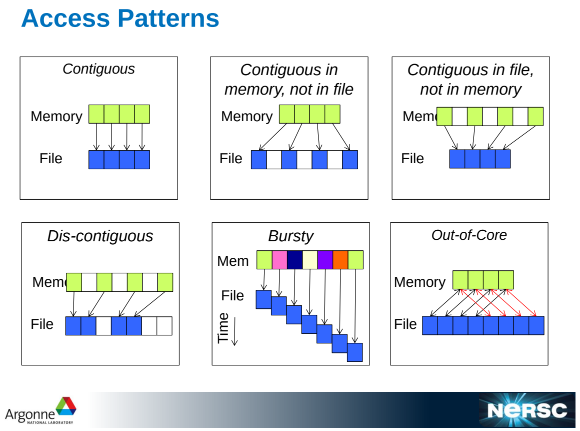
Serial I/O

Parallel Multi-file I/O

Parallel Shared-file I/O

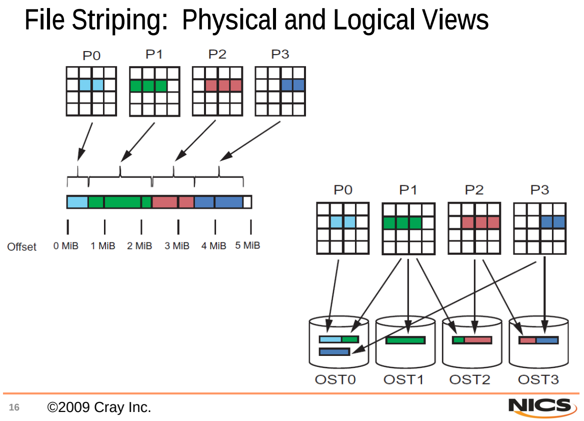
LR: tut145s3

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



LR: tut145s3

File Striping


File Striping: Physical and Logical Views

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.
- LR: Opportunity for doing better science (analysis) when have access to full spatiotemporal resolution data.

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

Parallel I/O

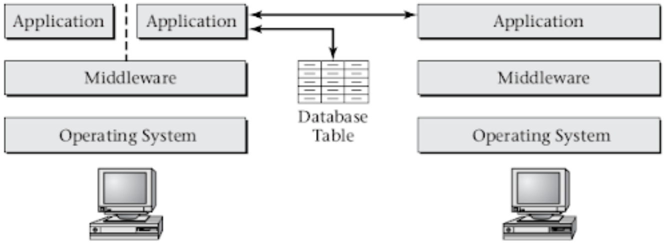
Data-driven Research

- **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

Middleware

- Middleware is software occupying a middle position between application programs and operating systems.LR: What is this database table in the picture? Just a common database for the application?



- Common middleware examples include relational database systems, application server middleware, message-oriented middleware, web middleware, and transaction-processing monitors.LR: Famous "brand" examples? I couldn't find any...

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.LR: Link ESDM here!
- 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?

LR: osm-rev1.3.1 1.3

Describe the role of middleware and file formats

- File formats

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,LR: Link ESDM here!
  - 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 Pop Quiz – Non-volatile Memory

1

2

3

4

5

6

7

8

9

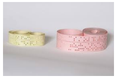
10

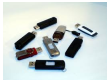
11

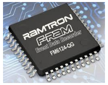
12

Hardware Pop Quiz – Non-volatile Memory

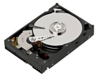
A	flash memory
B	read-only memory
C	ferroelectric RAM
D	hard disk drives
E	floppy disks
F	magnetic tape
G	optical discs
H	paper tape
I	punched cards

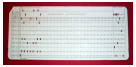

1

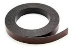

2

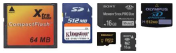


3

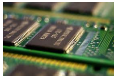

4

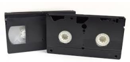

5

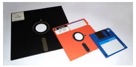

6

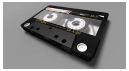


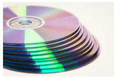
7


8


9


10


11


12

Hardware Pop Quiz – Non-volatile Memory

A	flash memory – 2/7
B	read-only memory – 8
C	ferroelectric RAM – 3
D	hard disk drives – 4
E	floppy disks – 10
F	magnetic tape – 6/9/11
G	optical discs – 12
H	paper tape – 1
I	punched cards – 5

1

2

3

4

5

6

7

8

9

10

11

12

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

- 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, Dax, and Damaris/Viz 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.

Outline

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

PIOSIMHD – THE MPI-IO SIMULATOR

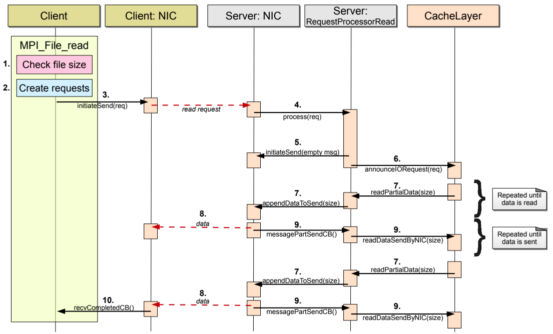
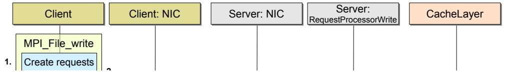


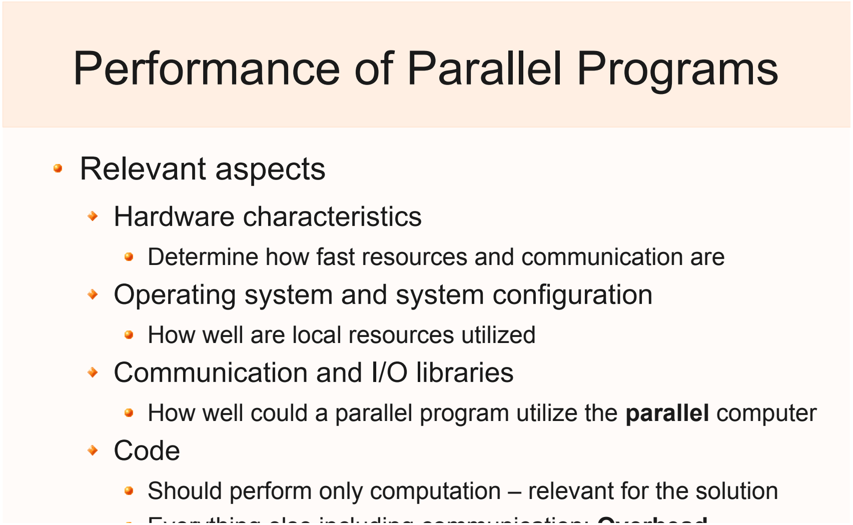
Figure 5.15.: Illustration of the client-server communication protocol – read path.



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.

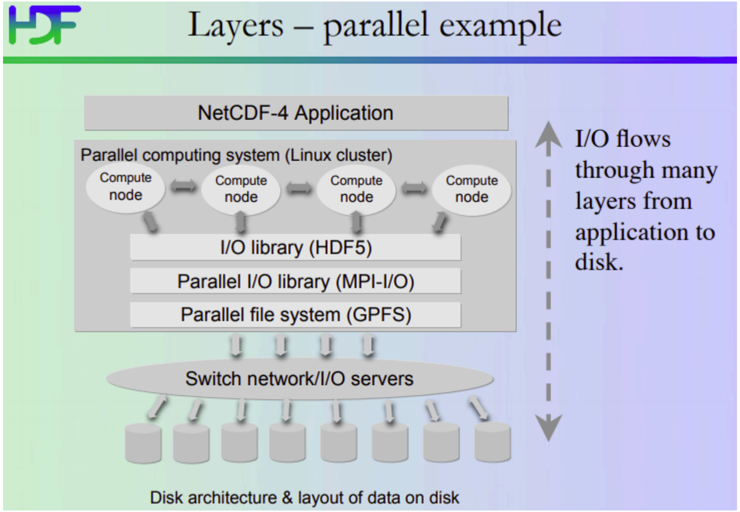
I/O Performance



Performance of Parallel Programs

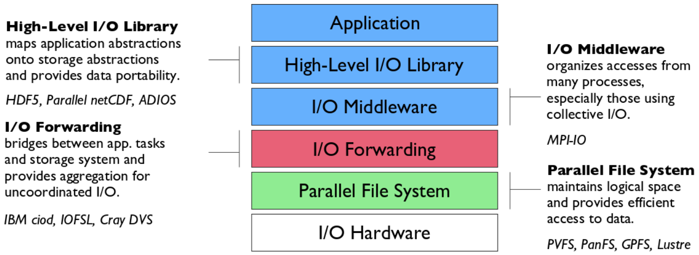
- Relevant aspects
  - Hardware characteristics
    - Determine how fast resources and communication are
  - Operating system and system configuration
    - How well are local resources utilized
  - Communication and I/O libraries
    - How well could a parallel program utilize the **parallel** computer
  - Code
    - Should perform only computation – relevant for the solution
    - Everything else including communication: **Overhead**

I/O Layers LR: (Stack?)


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.

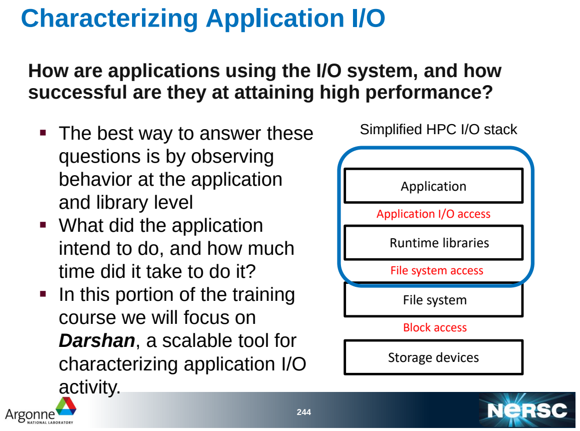
MPI-IO

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

PVFS, PanFS, GPFS, Lustre

Additional I/O software provides improved performance and usability over directly accessing the parallel file system. Reduces or (ideally) eliminates need for optimization in application codes.

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

Application

Application I/O access

Runtime libraries

File system access

File system

Block access

Storage devices

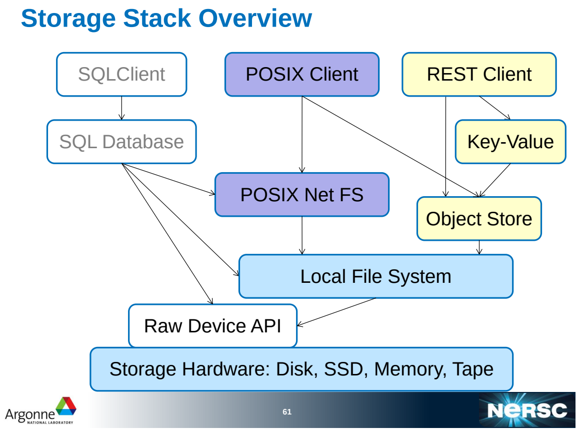
Argonne NATIONAL LABORATORY

244

NERSC

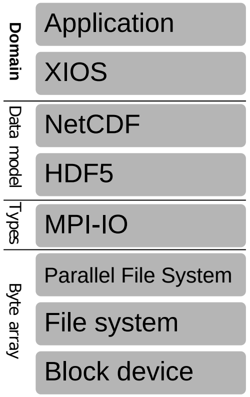
LR: tut145s3

Storage Stack

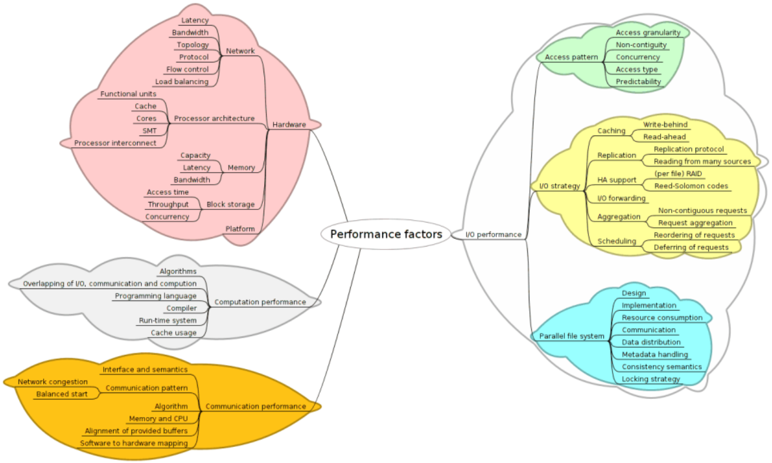


LR: tut145s3

Storage Stack



Typical Performance Issues



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:

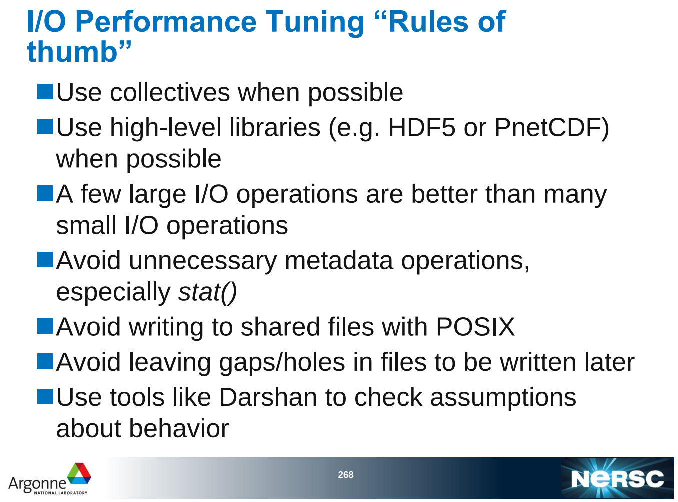
- Design
- Implementation
- Resource consumption
- Communication
- Data distribution
- Metadata handling
- Consistency semantics
- Locking strategy

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?

LR: skill-tree

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

268

I/O Performance Model

Big Data Cluster Characteristics

University of Reading

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

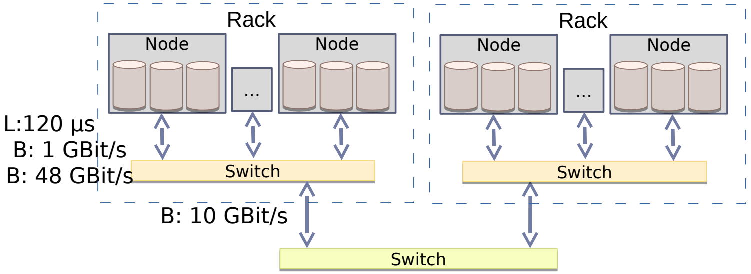


Figure: Architecture of a typical big data cluster

LR: lecture 5

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.

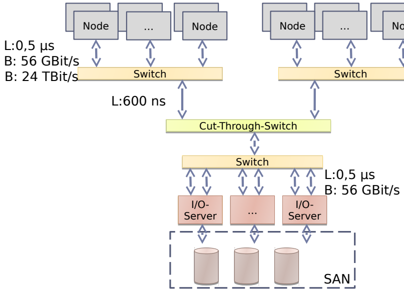
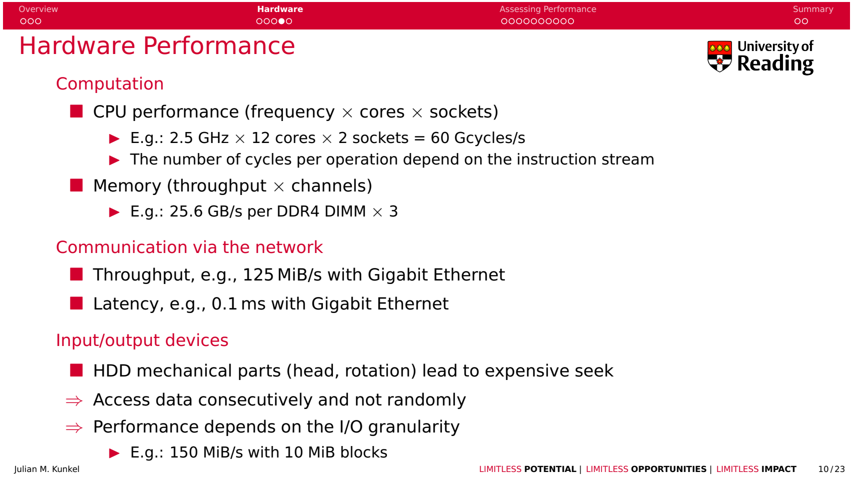


Figure: Architecture of a typical HPC cluster (here fat-tree network topology)

LR: lecture 5

I/O Performance Model



Hardware Performance

University of Reading

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

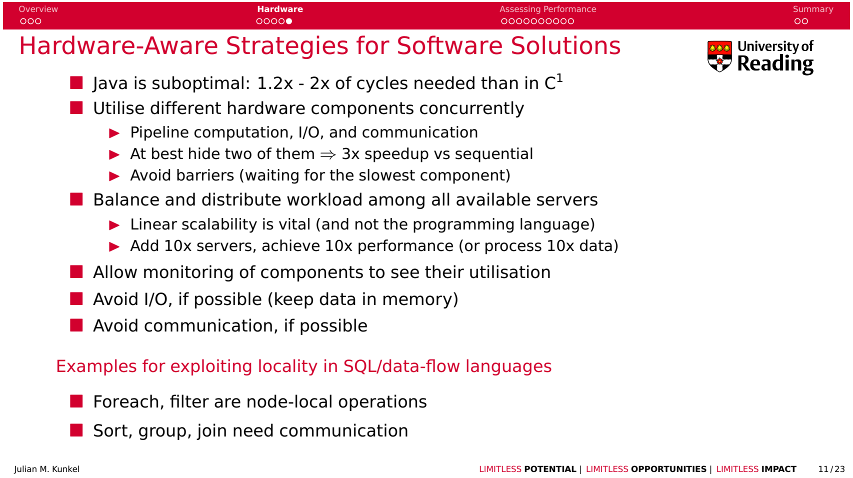
⇒ Access data consecutively and not randomly

⇒ Performance depends on the I/O granularity

  - E.g.: 150 MiB/s with 10 MiB blocks

LR: lecture 5

I/O Performance Model



Hardware-Aware Strategies for Software Solutions

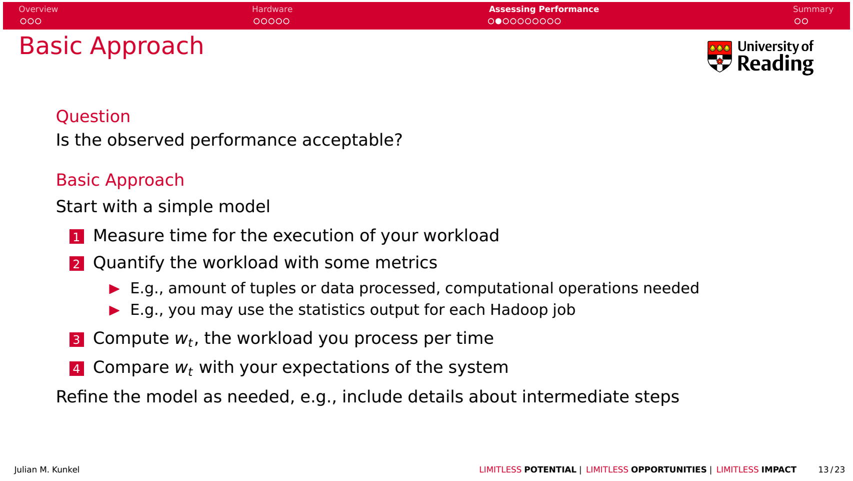
- Java is suboptimal: 1.2x - 2x of cycles needed than in C[1]
- Utilise different hardware components concurrently
  - Pipeline computation, I/O, and communication
  - At best hide two of them $\Rightarrow$ 3x speedup vs sequential
  - Avoid barriers (waiting for the slowest component)
- Balance and 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

LR: lecture 5

I/O Performance Model



Basic Approach

Question

Is the observed performance acceptable?

Basic Approach

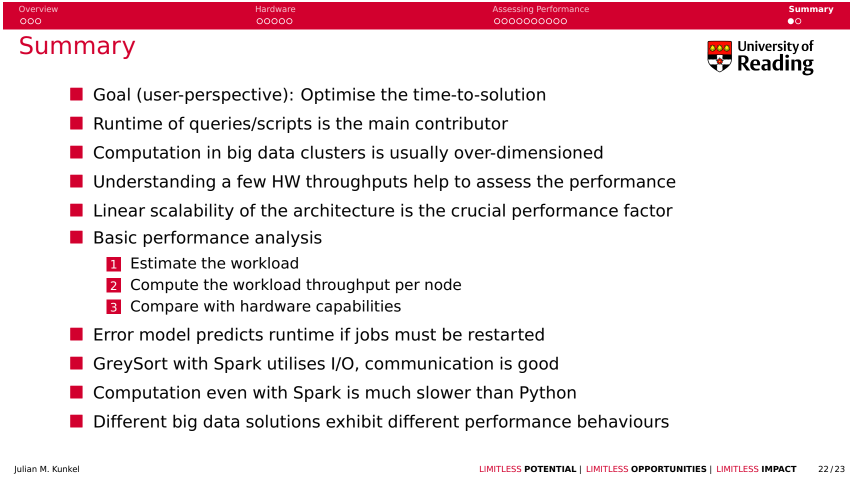
Start with a simple model

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

LR: lecture 5

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

LR: lecture 5

Outline

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.

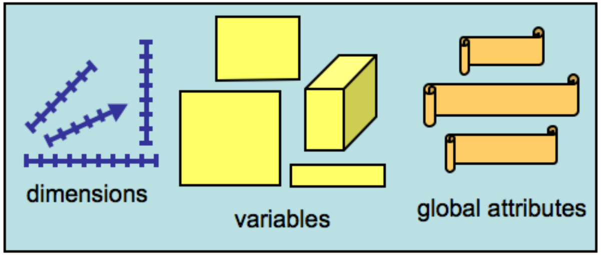
```
netcdf short {
dimensions:
  latitude = 3 ;
  longitude = 2 ;
variables:
  float sfc_temp(latitude, longitude) ;
    sfc_temp:units = "celsius" ;
data:

 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.

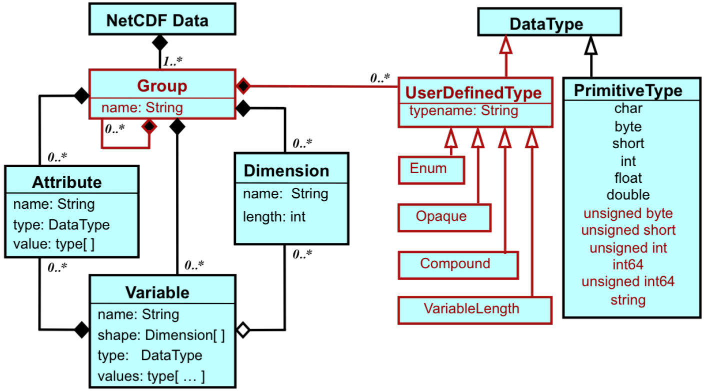
The Classic NetCDF Model

- 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

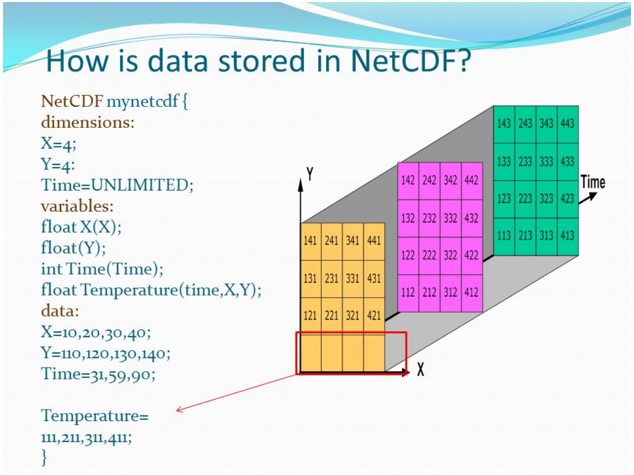
- A file has a top-level unnamed group.
- Each group may contain one or more named subgroups, user-defined types, variables, dimensions, and attributes.
- Variables also have attributes.
- Variables may share dimensions, indicating a common grid.
- One or more dimensions may be of unlimited length.

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?

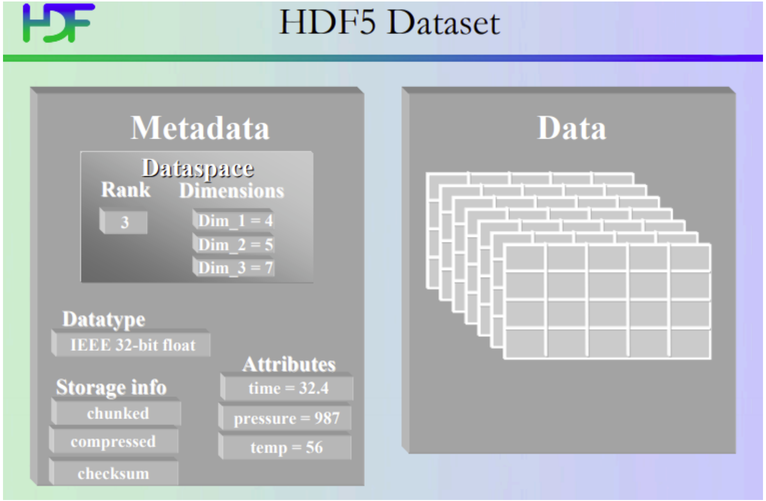
```
NetCDF mynetcdf {
dimensions:
X=4;
Y=4:
Time=UNLIMITED;
variables:
float X(X);
float(Y);
int Time(Time);
float Temperature(time,X,Y);
data:
X=10,20,30,40;
Y=110,120,130,140;
Time=31,59,90;

Temperature=
111,211,311,411;
}
```

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

LR: Sangireddy

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.
- 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

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**

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