I/O Perspectives for Earthsystem Data

Julian M. Kunkel

kunkel@dkrz.de

German Climate Computing Center (DKRZ), Research Group/Scientific Computing (WR)

05-11-2016

Outline

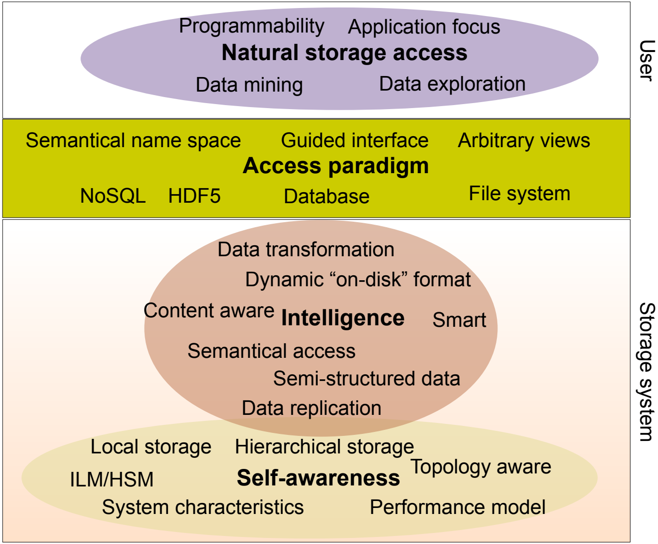
Features of an Optimal I/O Model

- Standardized interface
  - Portability, long-term support ...
- Close to application domain
  - Such as e.g. CDI; or lower level HDF5
- Performance portability
  - Utilizes network bandwidth (6.2 GiB/s per node on Mistral)
    - With network blocking 4:1, may become 1.6 GiB/s
  - Scalable with the number of nodes used
- Manages a wide range of different storage technologies
  - NVRAM, SSDs, HDDs, Tape
- Provides support to process Workflows (integration in Slurm)
- Each process can start I/O independently

Things a Developer/User Should not Care About

- Technical details/hints
  - Layout this variable with a chunk size of x:y:z
  - How many stripes, stripe sizes for Lustre
- File format for intermediate storage
  - As long as tools can access the data
  - You care when downloading data / pushing it on long-term storage
- About naming conventions for complex directory structures
  - Are you doing this still for your MP3 collection?
  - Lightweight databases can do much better

Personal Vision of Future Storage Systems



Peak Performance with Lustre @ DKRZ

Phase 1

- 29 SSUs · (2 OSS/SSU + 2 JBODs/SSU) = 58 OSS and 116 OSTs
- 1 Infiniband FDR-14: 6 GiB/s ⇒ 348 GiB/s
- 1 ClusterStor9000 (CPU + 6 GBit SAS): 5.4 GiB/s ⇒ **313 GiB/s**

Phase 2

Similar to Phase 1, performance adds up!

Performance Results from Acceptance Tests

- Throughput measured with IOR
  - Buffer size 2000000 (unaligned)
  - 84 OSTs (Peak: 227 GiB/s)
  - 168 client nodes, 6 procs per node

Type	Read	Write	Write rel. to peak
POSIX, independent[1]	160 GB/s	157 GB/s	70%
MPI-IO, shared[2]	52 GB/s	41 GB/s	18%
PNetCDF, shared	81 GB/s	38 GB/s	17%
HDF5, shared file	23 GB/s	24 GB/s	**10%**
POSIX, single stream	1.1 GB/s	1.05 GB/s	0.5%

- Metadata measured with a load using Parabench: 80 kOPs/s
- 25 kOP/s for the root MDS and 15 kOP/s for each DNE MDS

[1] 1 stripe per file

[2] 84 stripes per file on 21 SSUs

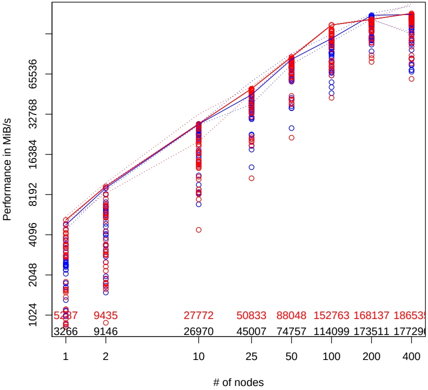
Observations to Take Away

- Single stream performance is much lower than on Blizzard
- Multiple threads need to participate in the I/O
  - 12 to 16 are able to (almost) utilize Infiniband
- Independent I/O to independent files is faster
- An optimized file format is important for fast I/O
  - e.g. NetCDF4/HDF5 achieves $< 1/2$ performance of PNetCDF
- Benchmarking shows sensitivity to proper configuration
  - 4x improvement is often easy to achieve

$\Rightarrow$ Let us vary the thread count (PPN), stripe count and node count

Current Situation with Lustre

- Measurement of synchronous **independent I/O** are promising
- IOR, **indep. files**, 10 MiB blocks
  - Measured on the production system
  - Slowest client stalls others
  - Proc per node: 1,2,4,6,8,12,16
  - Stripes: 1,2,4,16,116
- 120 GB/s on 100 nodes = 1.2 GB/s
  - 75% network peak (with conflicts in static routes)

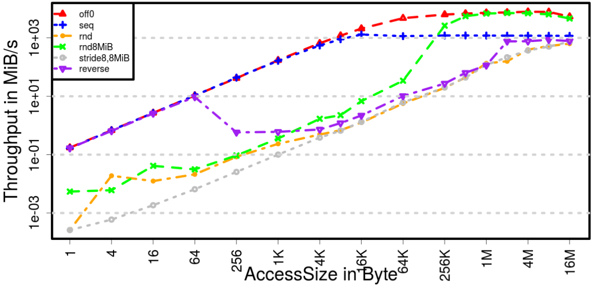


Best settings for read (excerpt)

Nodes	PPN	Stripe	W1	W2	W3	R1	R2	R3	Avg. Write	Avg. Read	WNode
1	6	1	3636	3685	1034	4448	5106	5016	2785	4857	2785
2	6	1	6988	4055	6807	8864	9077	9585	5950	9175	2975
10	16	2	16135	24697	17372	27717	27804	27181	19401	27567	1940

File Formats

- NetCDF4/HDF5 achieves about 10% peak I/O on Mistral
- POSIX achieves about 70% peak I/O on Mistral
- 7x improvement possible
- With suboptimal access pattern only 1/10th of HDF5 performance observable
- Sequential I/O is best, avoid random I/O (as expected)
    - Multi-threaded I/O looks like random I/O to Lustre



Read performance (clean cache)

Storage Technology

- Many upcoming technologies will be shipped until 2020
- NVRAM is **byte addressable**
- Located across the hierarchy, node-local storage, burst-buffers
- 3D-Xpoint via PCIe or as DIMM announced[3]
- Phase change memory for RMA[4]

	Memristor	PCM	STT-RAM	DRAM	Flash	HD
Chip area per bit (F^2)	4	8–16	14–64	6–8	4–8	n/a
Energy per bit (pJ)[2]	0.1–3	2–100	0.1–1	2–4	10^1–10^4	10^6–10^7
Read time (ns)	<10	20–70	10–30	10–50	25,000	5–8x10^6
Write time (ns)	20–30	50–500	13–95	10–50	200,000	5–8x10^6
Retention	>10 years	<10 years	Weeks	<Second	~10 years	~10 years
Endurance (cycles)	~10^{12}	10^7–10^8	10^{15}	>10^{17}	10^3–10^6	10^{15} ?
3D capability	Yes	No	No	No	Yes	n/a

5

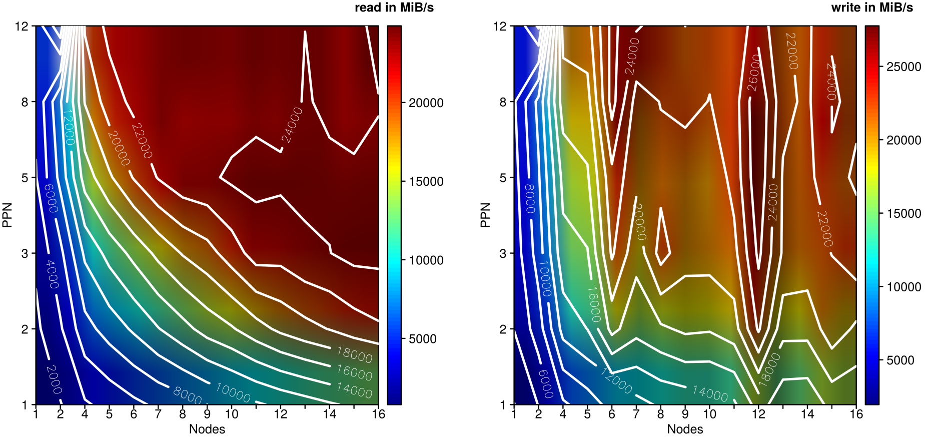
3 http://www.cnet.de/88154527/neue-speichertechnik-3d-xpoint-tausendmal-schneller-als-flash/

4 http://www.anandtech.com/show/9529/hgst-and-mellanox-show-off-san-fabric-backed-by-phase-change-memory

In-Memory Storage

- DRAM storage backed up on NVRAM (if needed)
  - Low variability, sequential vs. random does not matter
  - However, usually capacity is low (e.g., 6 TB)
  - Example: eXPress Disk (XPD) from Kove
  - Provides an API to read/write data to shared storage
- Conducted a evaluation on a test-system with XPD
  - Created an MPI-IO driver utilizing the system
  - Benchmark: I/O to a "shared" file using IOR; read/write similar
  - HDF5 tests are ongoing
  - Random I/O and sequential I/O are expected to be similar

Results of the MPI-IO Benchmark



(a) Read 16 KiB (b) Write 1 MiB

MPI-IO performance

Alternatives for Achieving Optimal I/O

- The file system and scientific format deliver wire-speed
  - Unfortunately, **not the case in production** anytime soon
  - Only possible for a coarse access granularity (e.g. 10 MiB)
- Burst-buffers (in the File system, middleware etc.)
  - Dedicated nodes available to all applications
  - Utilizes non-volatile storage (SSD, NVRAM)
  - Additional benefit: also useful for random access patterns
- Application-specific I/O servers
  - Allocate additional nodes/memory for caching results
  - Like a burst buffer
  - May also conduct some transformations (e.g. XIOS)
  - Problem: too many variants exist, performance portability

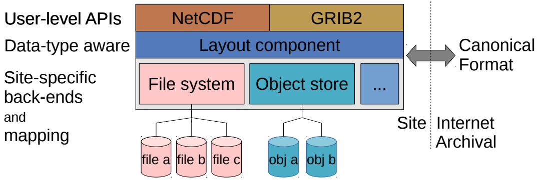
Dealing with Storage in ESiWACE

H2020 project: ESiWACE Center of Excellence

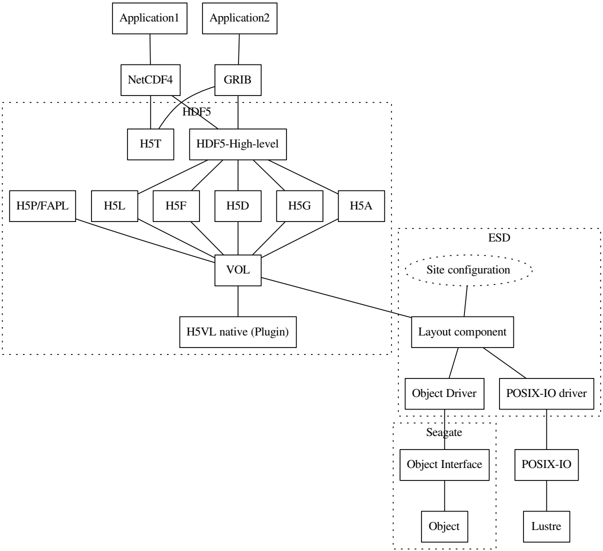
Work package 4

Partners: DKRZ, STFC, ECMWF, CMCC, Seagate, (HDFGroup)

1. Modelling costs for storage methods and understanding these
2. Modelling tape archives and costs
3. Focus: Flexible disk storage layouts for earth system data
   - Reduce penalties of „shared“ file access
   - Site-specific data mapping but simplify import/export
   - Allow access to the same data from multiple high-level APIs



Current Design



Design Aspects

- Layout component decides about mapping: logical $\Rightarrow$ physical
  - Utilizes multiple backends
  - Self-awareness: Decision based on hw characteristics and needs
  - Place scientific metadata in database
  - Allow to query metadata
  - On Mistral: use multiple files to store data
  - Redundant copies may use different layouts (serialization order)
  - User may give hints about high-level access pattern (intend)
- Export/Import data to standard format for data exchange

Strategic Considerations

- POSIX semantics will remain to be a performance burden
- Novel (NVRAM) technology enforces us to change our thinking
- My goal: Reduce the spinning disks at DKRZ
  - WITHOUT slowdown and even faster!
  - Better to intelligently manage storage to provide more CPUs!
- Utilize more storage tiers
  - In-memory computation is suitable as burst-buffers
  - SSDs, NVRAM for intermediate random workloads (small files)
  - Tape for read-seldom and workflows in the future
- Requires to: integrate user workflow into storage/scheduling

Workflows; Co-Design MPI & DKRZ

- Time to re-think and define end-to-end user workflows
- Example: Storage stores in-situ visualization and in-transit post-processing
  - Application may register workflow as DAG with code-snippets from CDO (lightweight plugins)
  - Embed CDO workflows into storage system similar to local processing in big storage workflows
- Example: No explicit file system hierarchy; on demand based on access patterns of scientific variables
- We can do theoretic analysis and evaluate (cost-efficient) alternatives but need you to re-think workflows
- Important to think about this now, as we have to think about DKRZ's future storage architectures

Summary

- We have the opportunity to influence storage vendors and middleware
- How to utilize byte-addressability is an open question
- Cost-efficiency requires workflow, scheduler and application to play together
- With a cost-efficient strategy we can do more science
- The importance of a fixed on-disk-format for the initial data creation will fade away

Sync vs. Async

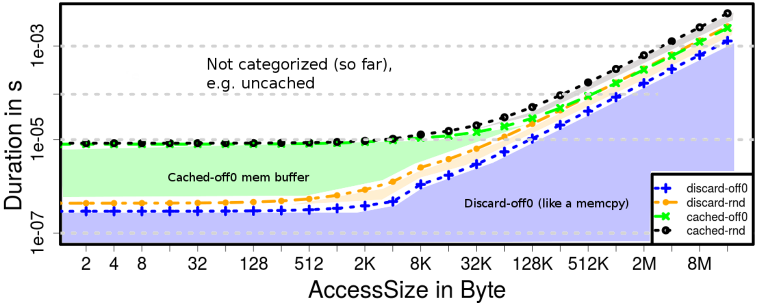
- Synchronous I/O is easy and very efficient model
  - With 64 GByte memory even in 10s/40s for large scale runs
- Async I/O
  - Cache data, requires at worst 2x memory (buy 2x memory? take it if available)
  - Competes with application's requirements (memory, network, evtl. CPU)
  - At very best 2x speedup, realistically much less

Async: RDMA pull vs. push

- Pull from servers requires net. QoS to avoid interference with app
- Push allows finer control (prevent I/O while communication) e.g. shown via ADIOS
  - Drawback: requires additional threads on the client to drive I/O

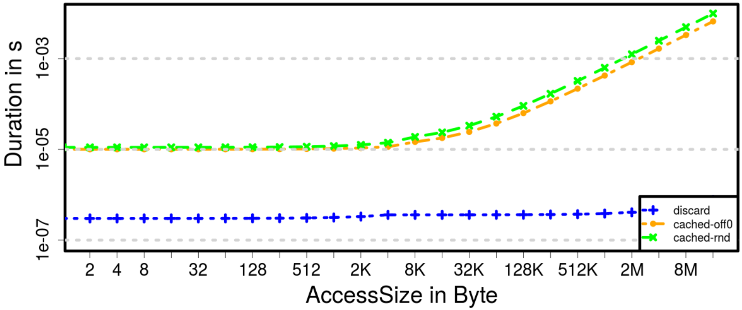
I/O Duration with Variable Block Granularity

- Performance of a single thread with sequential access
- Two configurations: discard (/dev/zero or null) or cached
- Two memory layouts: random (rnd) or re-use of a buffer (off0)



Read

I/O Duration with Variable Block Granularity

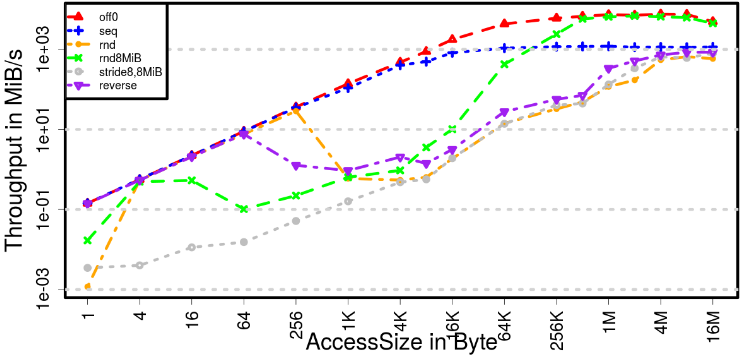


Write

- Memory layout has a minor impact on performance

⇒ In the following, we'll analyze only accesses from one buffer

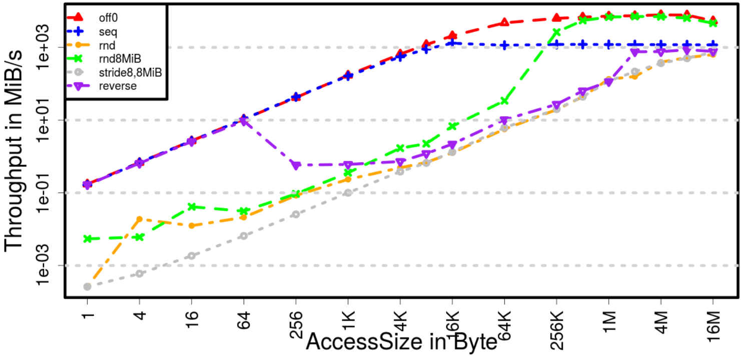
Throughput with Variable Granularity



Read – cached data

- Caching (of larger files, here 10 GiB) does not work
- Sequential read with 16 KiB already achieves great throughput
- Reverse and random reads suffer with a small granularity

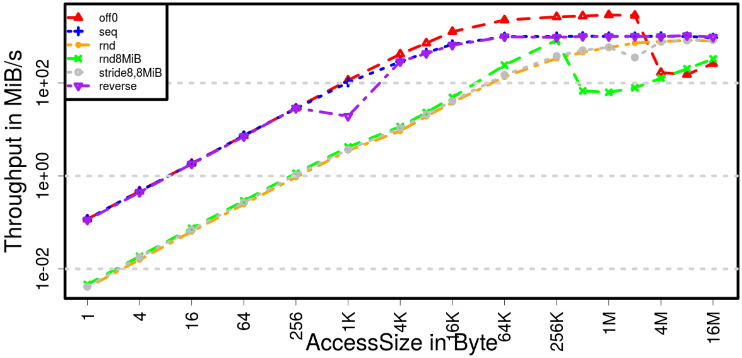
Throughput with Variable Granularity



Read – clean cache

- Read cache is not used
  - Except for accesses below 256 bytes (compare to the prev. fig.)

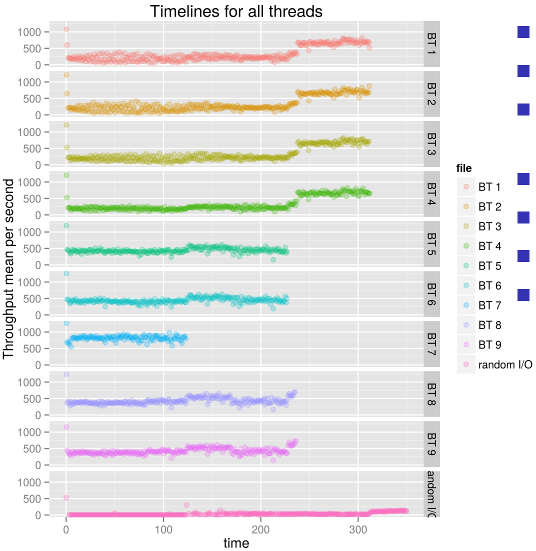
Throughput with Variable Granularity



Write

- Writes of 64 KiB achieve already great performance
- Reverse file access does not matter
- Abnormal slow behavior when overwriting data with large accesses (off0, rnd8MiB)

(Unfair) Sharing of Performance



- Storage == shared resource
- Independent file I/O on one OST
- Running 9 seq. writers concurrently (10 MiB blocks)
- One random writer (1 MiB blocks)
- Each client accesses 1 stripe
- Each client runs on its own node
- Observations
  - BT: 3 performance classes
  - RND without background threads: 220 MiB/s
  - RND with 9 background threads: 6 MiB/s
  - Slow I/O gets dominated by well-formed I/O

Performance Issues & Tunables

- I/O has to wait for the slowest server
  - A few slow servers significantly reduce IOR performance
  - Also: Congestion on IB routes degrade performance
- Interference between I/O and communication intense jobs
- Use a small number of stripes (for small files up to a few GiB)
  - On our system the default is 1
  - Create a new file with a fixed number: `lfs setstripe <file>`
  - Information: `lfs [getdirstripe|getstripe] <file|dir>`
- For highly parallel shared file access increase the striping
  - Performance is max. 5 GiB/s per stripe
- Avoid “ls -l”
  - It must query the size of all stripes from the OSTs
- Avoid moving data between different MDTs
- MPI Hints

Performance Issues & Tunables (2)

Changing Lustre's striping policy

```
# create two stripes with 10 MiB striping
$ lfs setstripe -c 2 -S $((1024*1024*10)) myfile
# query the information about myfile
# obidx shows the OST number
$ lfs getstripe myfile
myfile
lmm_stripe_count:   2
lmm_stripe_size:    10485760
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  6
  obdidx     objid     objid     group
       6       9258354      0x8d4572             0
      40       5927139      0x5a70e3             0
```

Performance Issues & Tunables (3)

MPI Hints

- Hints that have been proven useful during the acceptance test
- Collective access to shared files is useful for writes

```
# collective I/O
romio_cb_read = disable # serve each operation individually
romio_cb_write = enable # use two-phase I/O optimization

romio_no_indep_rw = false # can be true only if using collective I/O
# non-contiguous optimization: "data sieving"
romio_ds_read = disable # do not use data sieving
romio_ds_write = enable # may use data sieving to filter
romio_lustre_ds_in_coll = enable # may use ds in collective I/O
romio_lustre_co_ratio = 1 # Client to OST ratio, max one client per OST
direct_read = false # if true, bypass OS buffer cache
direct_write = false # if true, bypass OS buffer cache
```

Filesystem

Filesystem

- We have only one file system: /mnt/lustre01
- Symlinks: /work, /scratch, /home, ...
- However, each metadata server behaves like a file system

Assignment of MDTs to Directories

- In the current version, directories must be assigned to MDTs
  - /home/* on MDT0
  - /work/[projects] are distributed across MDT1-4
  - /scratch/[a,b,g,k,m,u] are distributed across MDT1-4
- Data transfer between MDTs is currently slow (mv becomes cp)
- Lustre will be updated with a fix :-)

High-level Considerations (2)

Embedded application/MPI vs. integrated middelware/file system

Application-specific I/O servers like XIOS, CDI-PIO etc.

- Requires N additional I/O servers to achieve sustainable performance, i.e. full network performance
  - If these cannot used for post-processing/in-situ vis and in-transit analysis, a lot resources wasted
- Async model requires smaller number of I/O servers but keeping interference with application on a level is difficult
- Coupling of models using different I/O servers difficult to manage process placement
- Reuse between applications is limited
- Maintainability: Future performance-portability on stake
- One-sided communication in MPI is a problem

Many potential solutions exist (also in the file system domain)

I/O Challenges [6]

- Deeper storage hierarchies
- Increasing in scale
- Increasingly complex topologies
- New data-driven science workflows !

Some conclusions (from SSIO workshops)

- Data management has to change or be replaced
- We are going away from the file model towards containers
- New generation of I/O middleware and service for NEW programming abstractions and workflows
- Integrated metadata capabilities (self-aware storage)
  - Automatic provenance capture

[6] http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20150727/Ross_SSIO_Workshops-201527.pdf

Outline

ICOMEX (old project)

Goals: Optimization of computation and I/O for ICO models

Achievements for I/O within the project

- Benchmark imitating ICON I/O (icon-output)
- Analysis/Illustration of alternative I/O modes
  - Individiual vs. forwarding vs. parallel I/O
  - Async vs. sync
- Parallel I/O was prototyped in ICON
  - Achieved half wire-speed for parallel runs!
- Implementation of compression schemes
- Optimizations of I/O middleware
  - Cacheless NetCDF
  - HDF5 multifile

`http://wr.informatik.uni-hamburg.de/research/projects/icomex/start`

ICOMEX I/O optimizations for ICON

Parallel I/O

- Each process writes to its own file independently
- 10x speedup (1-2 GiB/s tp per node instead of 0.15 GiB/s)
- Problem: File format changed, user experience limited

HDF5 multifile

- HDF5 I/O layer splits logical HDF5 file into subfiles
  - The original file became a directory (until reconstruction)
- Provides a coherent view to the data, log-structured
- Additional file for metadata
- First reading of the file reconstructs full compatible HDF5
  - Usually, reading/post-processing is done from one process
  - Not time critical
- Yielded same performance as independent
  - Reconstruction about 500 MiB/s, time-to-solution much better!

Other projects

Intel Parallel Computing Center (for Lustre)

- Implementation of client-server compression
- Will also improve Lustre throughput for uncompressed I/O
- Testing of client-side extensions on Mistral planned!

AIMES

- DSL & I/O for ICO models
- User-defined/workflow oriented lossy compression
- Little bit of optimization for HDF5/NetCDF

Description of Tasks (According to the Contract)

- A: Analysis of current solutions and workflows concerning I/O bottlenecks
- S: Developing HPC software solutions for climate models
  - e.g. advance I/O middleware, extend or create useful I/O libraries
- L: Development of a long-term strategy for efficient I/O
  - Goal: Reduce re-coding effort and foster performance portability
- T: Development of a optimization strategy tuning hard- and software for the phase 2 system
  - Also assist configuration decisions for phase 2, e.g. two FS?

A benefit for climate applications esp. ICON is expected

Proposal for 2016 (1)

Goals for 2016

- Develop a thorough understanding of the status-quo
- Identify key workloads besides ICON to foster understanding of typical I/O patterns AND used I/O middleware

Duties inc. matching task and PMs

- AT. 2M: Include I/O statistics measurement in DKRZ monitoring
  - Understand and include capturing of Lustre stats into our DKRZ monitoring
  - From clients via /proc and from servers either by using lmt or also /proc
- LT. 1M: Supporting the evaluation of Burst Buffers (e.g. with Bull)
- AT. 2M: I/O Performance analysis and report for Mistral
  - Using IOR, flavors of the icon-output for NetCDF4/HDF5 and real ICON
  - Goal: Thoroughly understand how the I/O path works currently
- T. 1M: Maintaining best-practise (on the DKRZ user webpage)

Proposal for 2016 (2)

- AT. 2M: Analyze intermediate I/O results from the monitoring
  - Identify problematic statistics and applications
  - Develop simple tool to automatically assess the results
- AT. 2M: Analyze and derive benchmarks from problematic applications
  - Using SIOX to analyze patterns
  - Extract and create benchmarks for good and very bad patterns of apps that still will be used in the future
  - Evtl. integrate them into the regression suite
- AT. 1M: Paper or Poster about the thorough performance analysis on our system: Performance issues for climate applications
- 1M: Slack for PhD, some teaching, communication with e.g. HDF5 group, participating in conferences, workshops

Some Synergies Between Projects

- AtosCoop[a] provide information about Mistral's behavior to other projects
- I/O insight from ESiWACE, IPCCL and AIMES is given back to Atos
- ESiWACE evolves HDF5 to deliver best performance on Mistral
  - Realizes: S. Developing HPC software solutions for climate models
  - Also create a library that can be useful outside of HDF5
- AtosCoop can evaluate benefit of client-side compression for Lustre

[a] Means this collaboration.

Requirements to BULL

- Contact for discussion of performance results
- Contact to Bull's own Lustre developer?
- Optional: Enable testing of I/O code on alternative (non-Seagate) systems
    - Either providing access to the systems OR conducting the well described test

System perspective

- Announce: remember memory block and start data transfer
- iteration_start(): permit data modification again, choosable semantics
  - Pause I/O – synchronous – Version 1
  - Memcpy (expensive)
  - Initiate COW for memory blocks that were not transferred
    - Small costs for changing page table entries in MMU
    - May result in a full copy of data but without memcpy()
  - Lazy, i.e. just transfer data from memory (may have changed)
- Data transfer: direct data transfer from memory to file system (no pc)
- Use QoS to prevent data transfer to disturb application communication
- Provides active-storage hooks for in-transit data-processing
  - Plugins can be executed somewhere in the I/O path
  - System optimizes the placement
  - Allows in-situ visualization, in-transit transformations
- Can redirect I/O to node-local NVRAM storage, store metadata in DB

Potential Uptake in the Application Domain

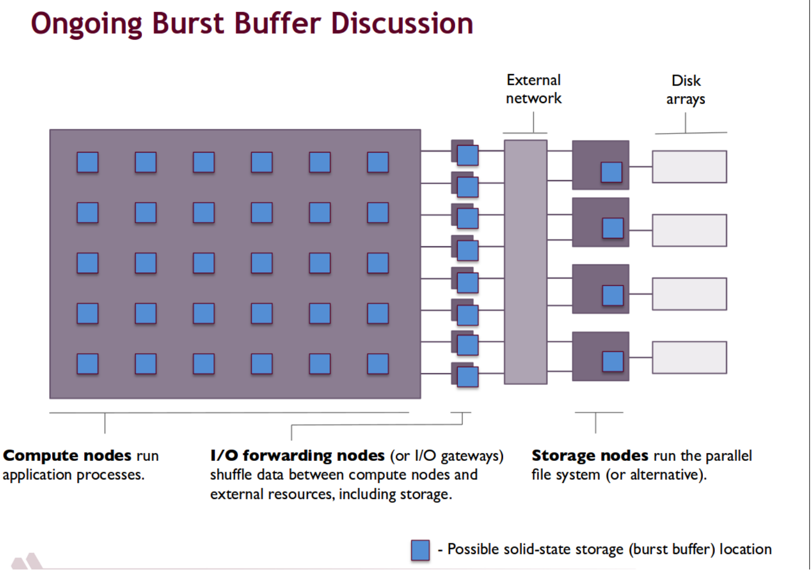
I/O Path

- Utilize new layer/middleware (ESiWACE)
- Integrate with HDF (ESiWACE) and CDI
  - Benefit of integration into HDF: Becomes a standard, increases maintainability
- CDI (Climate Data Interface from MPI-M), e.g. in use in ECHAM
  - CDI already offers a (similar) interface as discussed previously
  - Works already with ECHAM
  - MPI-M will use CDI more in the future

Active-storage hooks

- Synergies to CDO development
- Discussion with MPI-M started

Burst Buffers: Widely Accepted Strategy



In-situ and in-transit Analysis/Processing

- Data movement between CPU and storage is extremely costly
  - 5000x more than a DP FLOP[7]
  - 10 pJ per Flop (2018), 2000 pJ for DRAM access
- In-transit: analyze/post-process data while it is on the I/O path
- Some workflows are already in production: usually conducted in a cooperation between app/vis and storage team
- Examples
  - ADIOS [8]
  - DataSpaces [9]
- Burst-Buffers + In-transit processing capabilities: There won't be need for application-specific I/O solutions (XIOS, CDI-PIO, ...)

[7] http://www.fatih.edu.tr/ esma.yildirim/DIDC2014-workshop/DIDC-parashar.pdf

[8] Paper: Combining in-situ and in-transit processing to enable extreme-scale scientific analysis, 2012

[9] http://www.fatih.edu.tr/ esma.yildirim/DIDC2014-workshop/DIDC-parashar.pdf

Laboratory for I/O Investigation

Virtual Lab: Conduct what if analysis

- Design new optimizations
- Apply optimization to application w/o changing them
- Compute best-cases and estimate if changes pay off

So far: Flexible Event Imitation Engine for Parallel Workloads (feign)

- Helper functions: to pre-create environment, to analyze, ...
- A handful of mutators to alter behavior
- Adaption of SIOX is ongoing to allow on-line experimentation

Additional Research @ WR

- Compression of scientific data
  - Lossless (1.5:1 to 2.5:1)
  - Lossy: rate 12:1 to 50:1[10]
  - Interfaces for specifying tolerable loss
- Domain-specific languages
  - Retain code-structure
  - Improve readability
  - Intelligent re-structuring of code at compile time
- Alternative interfaces, usage of object storage
- Monitoring
- We push (computer science) standards towards your needs

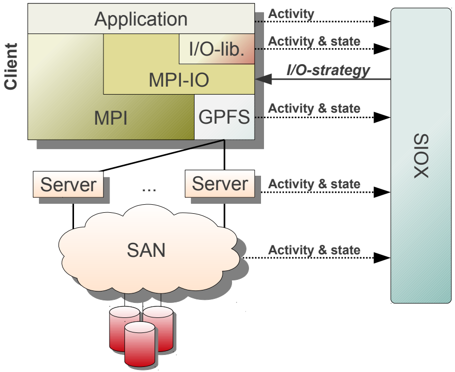
[10] WaveletCompressionTechniqueforHigh-ResolutionGlobalModelDataonanIcosahedralGrid,Wanget.al,2015

My Final 5 Cents

- Scientific productivity is the goal
- Future systems will change the way we use them for HPC
- We will be able to run legacy applications
  - Maybe at 5% what is possible with novel workflows
- Managing and accessing I/O will definitely change
  - Too many prototypes are already in production and more to come
- Standards across data centers are needed
  - Consortia to define and implement (storage, montioring etc.) APIs
- Need for separation of concern between domain scientists, scientific programmer, system architect and computer science
  - Increase the abstraction level, decouple code

Scalable I/O for Extreme Performance (SIOX)

Started as collaborative project between UHH, ZIH and HLRS



SIOX aims to

- collect and analyse
  - activity patterns and
  - performance metrics
- system-wide

In order to

- assess system performance
- locate and diagnose problem
- learn optimizations

SIOX Ongoing Work

Automatic assessing the quality of the I/O

```
Your Read I/O consisted of:
200 calls/100 MiB
10 calls/10 MiB were cached in the system's page cache
10 calls/20 MiB were cached on the server's cache
100 calls/40 MiB were dominated by average disk seek time (0.4
...
5 calls/100 KiB were unexpected slow (1.5s time loss)
```

Follow up Project

- Together with our partners we submitted a follow up project
- To increase scalability and assessment capability

Thinking About Optimal I/O

Assumptions

- With HDF5 modifications we'll achieve 95% wire-speed
- We consider periodic write and are starting the iteration BEFORE I/O is done

General application perspective (partly available today)

- Upon start: Register post-processing workflow as DAG with code-snippets (lightweight plugins)
  - e.g. code snippets from CDO
- Every iteration start is notified to I/O lib: iteration_start()
- Once a variable of a block is computed, "announce()" availability to I/O lib
  - Semantics: No modifications to this data until next iteration
- Communication phase is not degraded by I/O
- If insufficient memory is available, pause "announce()"