Challenges in Understanding, Optimizing and Procuring Storage Systems

Julian M. Kunkel

kunkel@dkrz.de

German Climate Computing Center (DKRZ)

09-05-2014

Outline

Challenges in Improving I/O Performance

- Users deal with different access styles
  - POSIX, (No)-SQL, HDF5, ADIOS,...
  - User-specific interfaces
- User workflow is typically unknown
- Storage systems are complex
  - Involves many hardware and software layers
  - Non-linear behavior caused by HW and SW characteristics
  - Complete understanding of full stack is barely possible
- Replicated features across the I/O stack
  - Locking / synchronization, Caching, read-ahead, ...
- Many tunable parameters
  - Interplay between different optimizations hard to predict
- Large volume & costs asks for data reduction techniques
- Zoo of "Solutions"
  - ADIOS, PLFS, SIOnlib, ...

Consequences

- High variability in I/O performance
  - Sensitive to workloads
  - Shared ressource
- Often lack in performance portability
  - Library performance depends on the system
  - Tuning of all libraries required
- Non-trivial to analyze
  - Lack of tools
  - Performance prediction?
- Users try to decouple I/O from computation
  - Trend: Implementation of “application I/O servers”...

Challenges for I/O Benchmarking

- How can we determine relevant performance characteristics?
- Example: Existing metadata benchmarks are often ”optimistic”
  - Batched creates/lookups/deletes can be optimized well

The Blizzard Supercomputer

- Computation: 247 nodes
  - Microprocessors: 16 Power6 dual-core (total: 8448 cores)
  - Memory: 4 GByte per core
  - Interconnect: 4xDDR-Infiniband
- File systems: GPFS
  - Capacity: 7 Petabyte (ca. 7000 disks)
    - Split into work and scratch file systems
  - 12 I/O servers (same as compute nodes)
  - Max. throughput: 30 GByte/s
- Tape archive: HPSS
  - 6 Oracle/StorageTek SL8500 libaries (+)
    - More than 67,000 slots
  - Variety of tape cartriges/drives
  - 500 TB disk cache

Exclusively used for earth-system research

Performance Prediction

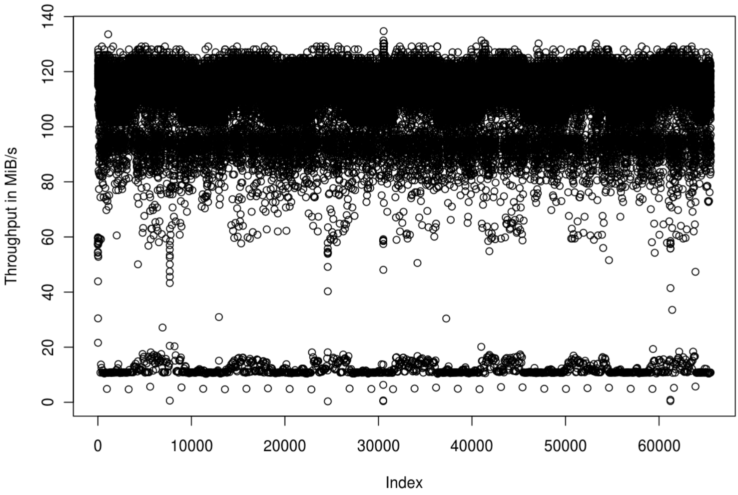
Simple models can be developed

- Number of clients, I/O servers participating (!)
- (Network, Block device) Latency
- (Network, block device) Throughput
- Distinction of cases: Non-cached, cached on servers, ...

Models are limited due to file system specific strategies: locking...

Predicting Performance Accurately is a Myth

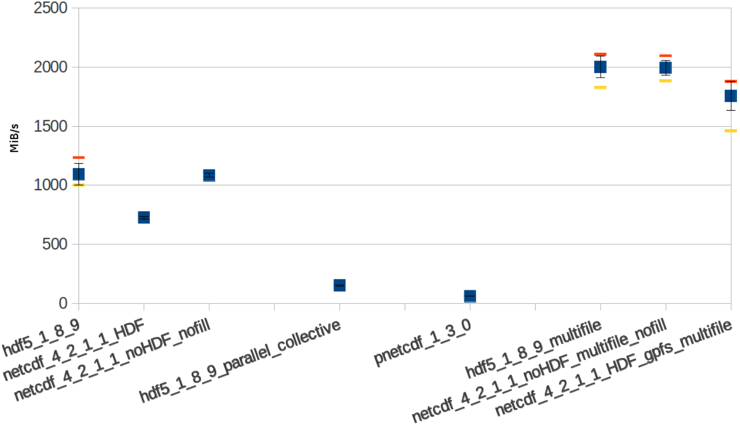
Already behavior of a local file system + one HDD is complex



Sequential write of 16 KiB blocks (*O_DIRECT*)

Variation of I/O Performance

- I/O access pattern of one climate application
- Performance of different library builds
- Measured on the Blizzard: One node, 32 procs
- Observation: Performance of individual files >> shared file



Challenge: Selecting the Appropriate Hints

- Example in MiB/s:

blizzard, IOR, 64 procs, 2 nodes				
	shared file		individual file	
	write	read	write	read
POSIX	710	3750	4300	5300
MPI-I/O	58	34	84	64
no datashipping			4836	5988

- Result of a (time-consuming) PMR with IBM:
  - Datashipping is the culprit (we believed from the beginning)
  - To switch it off, set environment variables
    - export MP_IOAGENT_CNT=all
    - Setting it to the number of procs behaves differently!
    - export MP_IO_BUFFER_SIZE=8M #for example
  - And hint within the application `IBM_largeblock_io=true`

Analyzing Usage

System monitoring

- Ganglia: 15 GByte/s sustained throughput
- `df -i`: 200 Million files.

User-space analysis

- (Workflow)
- Distribution of file sizes
- Distribution of file formats
- Potential of data-deduplication
- Potential of (lossy) compression

Approach

- Admin created a file list (using GPFS tools)
  - Covers 3.8 PByte of capacity and 155 Mio files
- Python script to analyze distribution and extensions

File formats

Motivation

- Gear optimization effort towards mostly used I/O libraries
- Understand the requirements for the procurement

Accuracy of the approach

- Many users use numerical extensions for created files
- 40% of small files have the extension "data" or "meta"
- Using `file` and cdo to determine file type crashed servers

Results

- NetCDF: 21 Million files (17% of files, 34% of capacity)
- Grib: 9 M files
- HDF5: 200 K files
- Tar: 12% capacity!

File size distribution

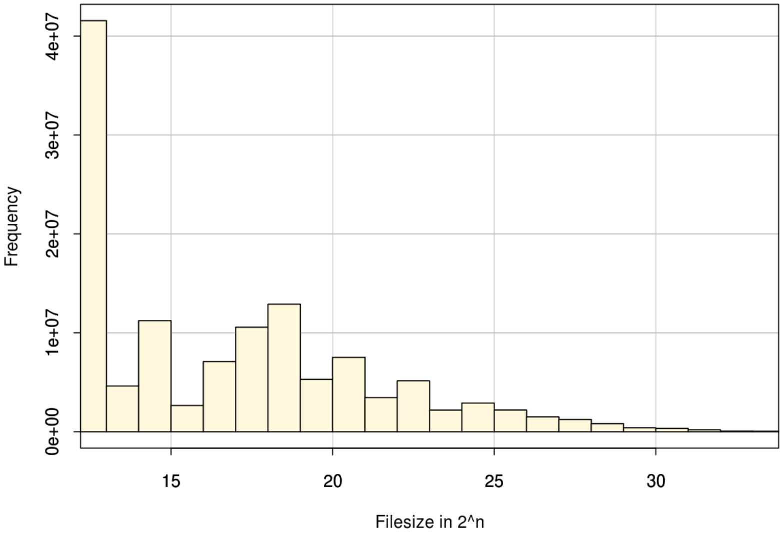
Results

- 28% of files $<$ 8 KiB
- 76% of files $<$ 1 MiB
- 90% of files $<$ 2 MiB
- 99.8% of files $<$ 1 GiB
- 6082 files $>$ 8 GiB

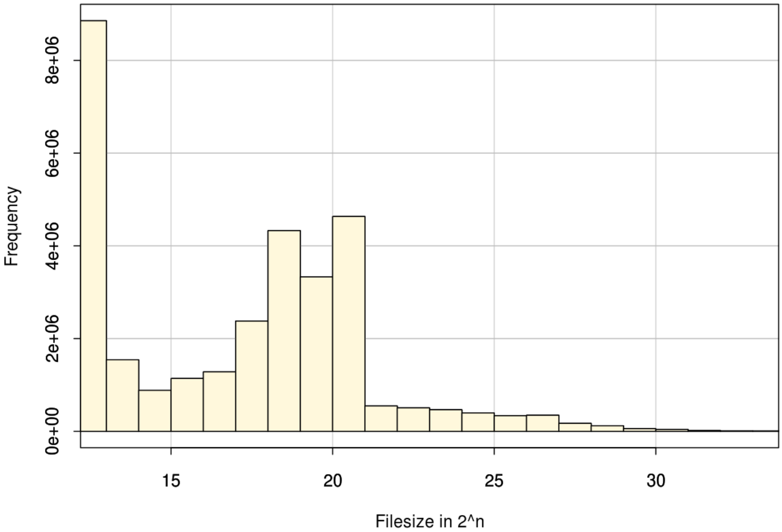
Conclusion

- 400 GByte SSD-tier could host all files $<$ 8 KiB...
- Avoid interference of small file access to parallel I/O

File size distribution: Work (ca. 110 Mio files)



File size distribution: Scratch (ca. 45 Mio files)



Trend: Application-specific I/O Servers

Since parallel I/O is slow users develop their own I/O middleware.
A workaround to "fix" performance issues in a broken architecture.

- Subset of processes dedicated for I/O
- Model ships data to "I/O servers"
- May perform additional data conversion (grid, reductions...)
- Examples: XIOS, CDI, ... ($>$ 4 in the climate community)

Challenges

- Adds another complex layer (not) easy to understand
- Performance portability
- Coupling of models with different styles of I/O servers
- Process mapping and parameterization

Data Deduplication

Approach

- Scanned 12 sets of home-directories independently (1 PB)
  - De-duplicate only within each set
- Tool used content-defined chunking to compute blocks

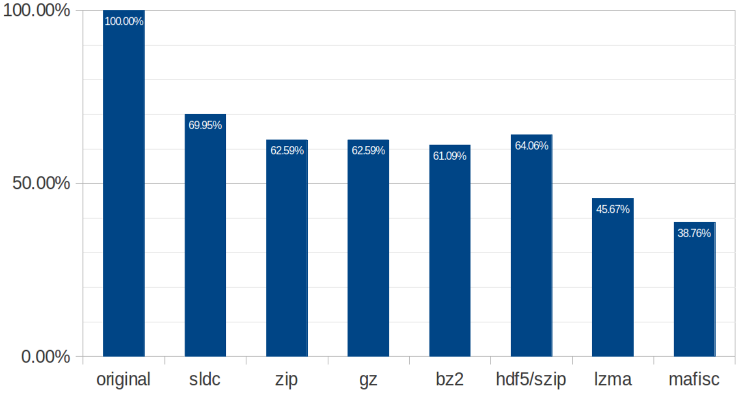
Results

- 20-30% of data is redundant
- Full file duplicates cost between 5-10%
- A small fraction of chunks is highly duplicated
  - Top 5% of chunks account for 35% capacity
  - Between 3-9% of data are zeros
- A lot redundancy is caused by tarballs (unpacked & archive)

Reference: *A Study on Data Deduplication in HPC Storage Systems. Meister et.al. SC'12.*

Lossless Compression

- Compressing a set of files from the CMIP5 experiment
- MAFISC is a set of filters that are applied before using e.g. LZMA
- Especially useful for long-term archival

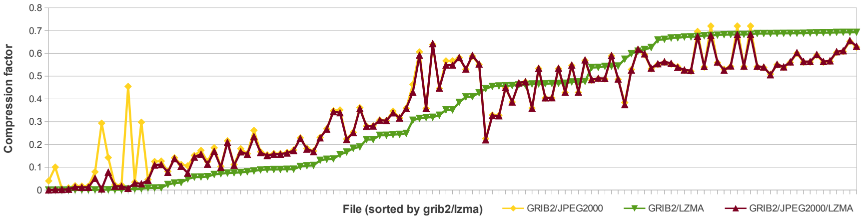


Reference: *Reducing the HPC-Datastorage Footprint with MAFISC - Multidimensional Adaptive Filtering Improved Scientific data Compression. Hübbe et.al, ISC'12*

Lossy Compression

Compressing 150 ECHAM 32 Bit float variables with GRIB 22 bit

- GRIB uses quantization to store a float as integer values
- Additionally JPEG2000 or LZMA lossless compression is applied
- Several synthetic 2D variables have been tested (< 10)



compression factor sorted by GRIB2/LZMA

Reference: *Evaluating Lossy Compression on Climate Data. Hübbe et.al, ISC'13*

Overview of Relevant Aspects for I/O

Selected I/O Requirements

- 45 Petabyte storage capacity (5 Billion files)
- Maximum number of file systems limited
- Short recovery time of the file system (power outages...)
- Minimal I/O throughput and metadata performance
- Optional: supply of a burst buffer with balanced characteristics

Benchmarks

- IOR: I/O throughput
- Parabench: Metadata performance
- Application benchmarks have a light-weight I/O footprint

IOR

Pattern

- Round-robin data access
- Blocksize: 2 MB (10^6 Byte)
- taskPerNodeOffset to read data on another node than written
- Arbitrary process numbers and nodes
- Testcase big enough to overrun caches, example data volume:
  - Access 3 x amount of memory per node
  - Access 3 x amount of cache of the I/O servers
- Layers: HDF5, PNetCDF, MPI (shared) and POSIX (individual)

IOR

Other rules

- Testcase must be big enough to overrun caches
  - We may upscale problem sizes during acceptance testing (if we believe we observe cache effects)
- Best value $min(read, write)$ for an arbitrary configuration
- Vendors are free to apply hints, modify libraries and parameters (but not the benchmark code)

Rationales

- Not aligned to file system blocks (typical application case)
- Show the need to commit for high-level I/O libraries
- Individual file access with POSIX is typically the best case
- Understanding performance loss in high-level I/O
- Selecting the better scientific I/O interface (HDF5 or PnetCDF)

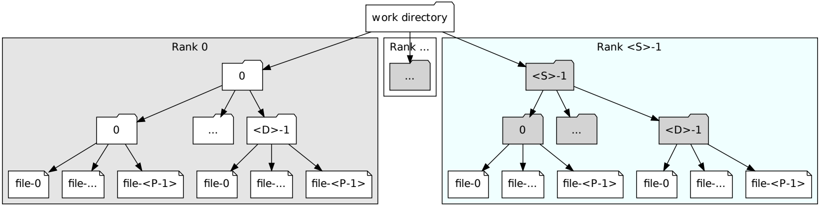
Parabench

- Flexible workload generator
  - Especially designed to mimic metadata workloads
  - Executes a script
- Uses R to visualize results

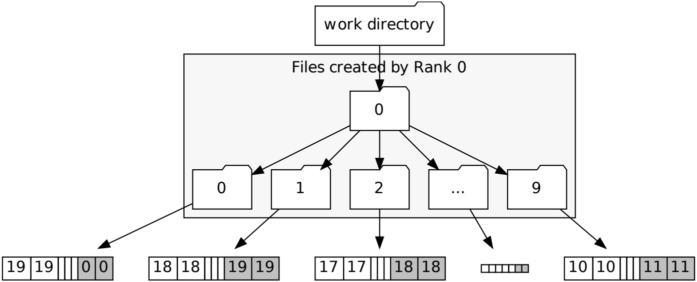
Benchmarking Script: *system-load-new*

- Mimic concurrent access to small files in a directory structure
  - Scalable to test realistic system sizes (e.g. 1000 users)
- Phases: Pre-creation, Load, Cleanup
  - Pre-creation:
    - Each process creates D directories x P files
  - Load (iterate across D directories x N files)

    Files in each folder are read by one process and written by another:
    - Create/write a new file in one folder
    - Stat an existing file, read it, delete it
- Files are e.g. 3900 Byte in size



Directory structure and pre-created files



Ranks accessing files of the directory structured created by Rank 0

P=5, N=2, size=20, D=10, O=1. Grey files are created during the load phase.

Example Operation for one Rank

File iterator	Directory iterator	Created	Accessed
0	0	0/0/file-5	2/0/file-0
0	1	2/1/file-5	4/1/file-0
0	2	4/2/file-5	6/2/file-0
0	3	6/3/file-5	8/3/file-0
0	4	8/4/file-5	10/4/file-0
0	5	10/5/file-5	12/5/file-0
0	6	12/6/file-5	14/6/file-0
0	7	14/7/file-5	16/7/file-0
0	8	16/8/file-5	18/8/file-0
0	9	18/9/file-5	0/9/file-0
1	0	0/0/file-6	2/0/file-1
1	1	2/1/file-6	4/1/file-1
1	2	4/2/file-6	6/2/file-1
1	3	6/3/file-6	8/3/file-1
1	4	8/4/file-6	10/4/file-1
1	5	10/5/file-6	12/5/file-1
1	6	12/6/file-6	14/6/file-1
1	7	14/7/file-6	16/7/file-1
1	8	16/8/file-6	18/8/file-1
1	9	18/9/file-6	0/9/file-1

Folders and files accessed by Rank 0 during execution
Pre-create=5, N=2, size=20, Dirs=10, Offset=2

Selection of (Preliminary) Experiments

Configuration

- Blizzard[1]: GPFS, 12 IO servers, 4x-DDR-Infiniband, HDDs only
- Lustre: 2.5, 10 IO servers, GigE
  - Intel 320 series SSD and WD20EARS HDDs

Testcase

- P=2917 D=10 N=1250 O=24
- Blizzard: nodes=10 procsPerNode=24 files=10000800
- Lustre: nodes=7 procsPerNode=12 procs=84 files=3500280

Observed performance

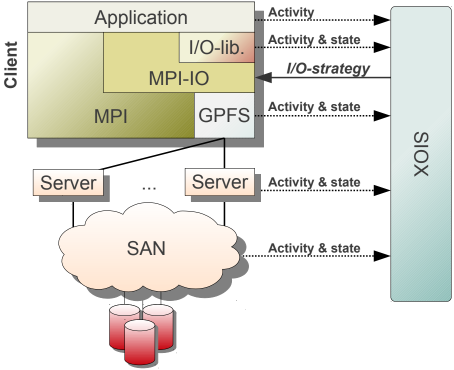
Phase	Blizzard	Lustre HDD	Lustre OST:HDD, MD:SSD	Lustre SSD
precreate [creates/s]	148565	19375	38108	75488
load [ops/s][2]	2950	3372	2510	21385
cleanup [deletes/s]	1250	1604	5852	6871

[1] In production, non-exclusive usage

[2] Each iteration consists of 4 operations: create/write, stat, read, delete

Scalable I/O for Extreme Performance (SIOX)

Collaborative project between UHH, ZIH, HLRS & (IBM)



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

Application-Specific I/O Workloads with Trace-Replay

Approach

- Extract application I/O captured in (SIOX) traces
- Replay them in an external tool

Flexible Event Imitation Engine for Parallel Workloads (feign)

- Helper functions: to pre-create environment, to analyze, ...
- Virtual Lab: Offer mutators to alter behavior
  - What if analysis, to evaluate benefit without changing apps
    - Design an optimization
    - May use lookahead to implement an oracle $\Rightarrow$ best-case
    - Apply it to arbitrary application traces
    - Replay modified traces
  - Interact with SIOX to explore parameter space

E10

- The Exascale I/O Initiative (former EIOW)
- Goal: Development of a Middleware with advanced features
  - Complete redesign of the I/O system
  - Different back-ends (hardware, file systems)
  - Arbitrary schemas (POSIX, HDF5, Flatland, ...)
  - Abandon restrictions of POSIX in the long run
  - Guided interfaces / Behavior indicators
  - Embedded monitoring & performance optimization
- International and open initiative
  - Collaboration: Xyratex, BSC, JGU Mainz, UHH, ...
  - Driven by the needs of the community (e.g. in requirement workshops)
  - Work-in-progress
- We will prepare a white-paper for ISC

Guided Interfaces

Guiding vs. automatism vs. technical hints

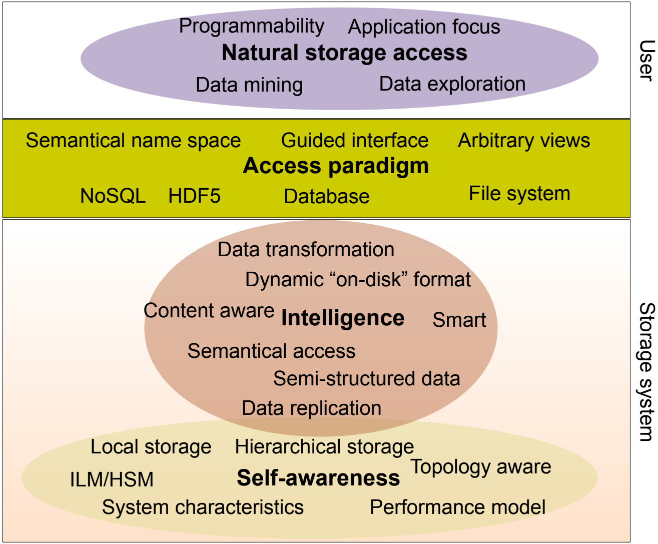
Users provide additional information to guide an intelligent system. The I/O stack exploits this information.

Information which could be provided by users

- Data types
- Semantics
- Relations between data
- Lifecycle (especially usage)

Several issues have already been addressed in different access paradigms. Also some behavioral hints exist: open() flags, fadvise()

Personal Vision of Future Storage Systems



Summary

- Better systematic analysis of users' workflow needed
- Analysis of the usage helps steering optimization effort

- Better interfaces and storage systems are needed
- To improve TCO, (lossy) compression will become mandatory

- Current procurement: IOR and Parabench MD benchmark
- Future: Trace-replay tool to mimic application behavior

- I/O has a long way to go: time to re-think from scratch