Understanding Burst Buffers with NetCDF-Bench

Eugen Betke, Julian Kunkel

Research Group
German Climate Computing Center

SIG-IO-UK Workshop
2018-06-06

Table Of Content

Introduction

- Capturing current performance values
  - How much performance can we achieve on our system?
  - Does performance changes over time?
- Understanding I/O performance
  - How different access patterns or features influence the I/O performance?
  - What are the most efficient settings?
- HPC procurement
  - Which technologies are beneficial for our applications / next generation HPC?

Climate data

Large scale climate applications can access **terabytes of data** on parallel file systems in a single run. Mostly, **NetCDF4** file format is used to access data.

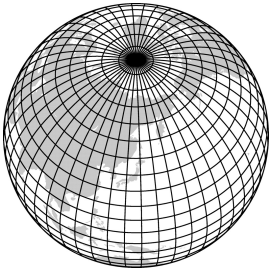


Figure: Regular grid [2] (legacy)

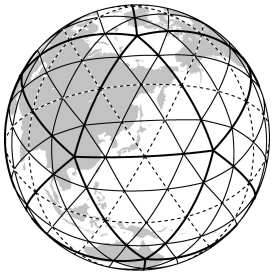


Figure: Icosahedral grid [2] (new)

Data arrangement in a file depends on earth grid used by application. Most large experiments on Mistral (HPC of DKRZ) are using regular and icosahedral grids.

Is there any benchmark, that can **approximate the behaviour** and provide a **realistic performance** values?

Under the hood, NetCDF4 uses HDF5. Maybe, IOR with HDF5-API can do the job.

Table Of Content

Running IOR with HDF5-API

```
mpiexec -n 2 ./ior -s 5 -t 5242880 -b 5242880 -o ./testfile.h5 -i 1 -v -k -f tests/HDF5
```

IOR example (**-s** - number of segments; **-b** - block size; **-t** - transfer size)

```
IOR START
   reorderTasksConstant=1
   taskPerNodeOffset=1
   api=HDF5
   RUN
IOR STOP
```

Configuration file

- **reorderTasksConstant** - reorders tasks by a constant node offset for writing/reading neighbor's data from different nodes [0=FALSE]
- **taskPerNodeOffset** - for read tests. Use with -C & -Z options. [1] With reorderTasks, constant N. With re-ordertasksrandom, >= N

Insight into HDF5 testfile

```
mpiexec -n 2 ./ior -s 5 -t 5242880 -b 5242880 -o ./testfile.h5 -i 1 -v -k -f tests/HDF5
```

```
~ h5dump -A testfile.h5
HDF5 "testfile.nc" {
  GROUP "/" {
    DATASET "Dataset-0000.0000" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( 1310720 ) / ( 1310720 ) }
    }
    DATASET "Dataset-0000.0001" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( 1310720 ) / ( 1310720 ) }
    }
    DATASET "Dataset-0000.0002" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( 1310720 ) / ( 1310720 ) }
    }
    DATASET "Dataset-0000.0003" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( 1310720 ) / ( 1310720 ) }
    }
    DATASET "Dataset-0000.0004" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( 1310720 ) / ( 1310720 ) }
    }
  }
}
```

IOR constraints

- Segments are stored in separate datasets
- transfer size = block size
- HDF5 features can not be tested

Conclusion

- No suitable access patterns, due to many metadata operations.
- Usually, transfer size in climate applicatons is smaller than block size. Data is written not as single block, but piecewise.

NetCDF Performance Benchmark Tool

- Developed at DKRZ
- Written in C
- Published on GitHub under LGPL License
  - `https://github.com/joobog/netcdf-bench`
- Mimics scientific data
  - Data is written in time steps
- Supports different NetCDF4 features
  - Compression, chunking, independent/collective I/O, fill value, unlimited dimensions

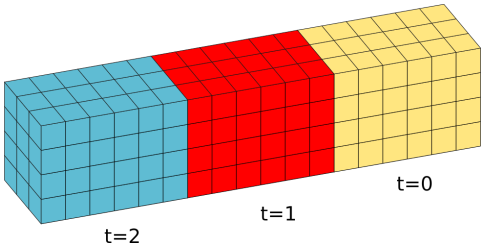


Figure: Data geometry (t:x:y:z)=(3:6:4:3)

NetCDF-Bench output

```
mpiexec -n 2 benchtool --nn=2 --ppn=1 -b=1:1024:256:5 -d=5:2048:256:5 -r -w -f=./testfile.nc
```

NetCDF-Bench example (**-d**: data geometry; **-b**: block geometry)

```
Benchtool (datatype: int)
Data geometry (t:x:y:z x sizeof(type))        5:2048:256:5 x 4 bytes
Block geometry (t:x:y:z x sizeof(type))       1:1024:256:5 x 4 bytes
Datasize                                          52428800 bytes              (52.4 MB)
Blocksize                                          5242880 bytes              (5.2 MB)
I/O Access                                     independent
Storage                                         contiguous
File length                                          fixed
File value                                              no
                                                       min                    avg                    max
benchmark:write    Open time                                0.1929476971           0.1929488905           0.1929500839 secs
benchmark:write    I/O time                                 0.0629276498           0.0629278460           0.0629280421 secs
benchmark:write    Close time                               0.0829069570           0.0829072705           0.0829075840 secs
benchmark:write    I/O Performance (w/o open/close)       794.5583288194         794.5608056386         794.5632824579 MiB/s
benchmark:write    I/O Performance                        147.5863619891         147.5866598368         147.5869576846 MiB/s
                                                       min                    avg                    max
benchmark:read     Open time                                0.0039127499           0.0039127710           0.0039127921 secs
benchmark:read     I/O time                                 0.0124143478           0.0124145120           0.0124146761 secs
benchmark:read     Close time                               0.0004221359           0.0004225459           0.0004229560 secs
benchmark:read     I/O Performance (w/o open/close)      4027.4912891123        4027.5445416068        4027.5977941012 MiB/s
benchmark:read     I/O Performance                       2985.0575402544        2985.1051135732        2985.1526868921 MiB/s
```

Insight into NetCDF-Bench test file

```
1 mpiexec -n 2 benchtool --nn=2 --ppn=1 -b=1:1024:256:5 -d=5:2048:256:5 -f=./testfile.nc
```

NetCDF-Bench example (**-d**: data geometry);**-b**: block geometry)

```
1  ~ ncdump -h testfile.nc
2  netcdf testfile {
3  dimensions:
4    time = 5 ;
5    dim_1 = 2048 ;
6    dim_2 = 256 ;
7    dim_3 = 5 ;
8  variables:
9    int data(time, dim_1, dim_2, dim_3) ;
10 }
```

NetCDF header

```
1 ~ h5ls testfile.nc
2 data                     Dataset {5, 2048, 256, 5}
3 dim_1                    Dataset {2048}
4 dim_2                    Dataset {256}
5 dim_3                    Dataset {5}
6 time                     Dataset {5}
```

Listing content with an HDF5 tool

Table Of Content

Kove® XPD® Specifications

- In-memory storage offers high troughput and low latency
- The Kove XPD is an example of a scalable in-memory storage system

Figure: Kove® XPD® L3 [1]

Capacity	64GB - 1.5TB per 1U unit, mesh connectable for larger capacities
Bandwidth	27+ GB/sec (12 μs latency and 100 ns variance), sustained
IOPS	43+ million (3.0 μs latency and 100 ns variance), sustained
Response Time	<2.5 μs, random, consistent, sustainable for any time duration
Data Protection	- Multiple modes for de-staging data from RAM onto persistent media
	- Integrated instant copying
	- Integrated monitoring logic for system health management and UPS

Storage Access

The XPD offers various storage access methods

- Block device
  - Can be used as ordinary block device, e.g., with EXT4
  - Needs (slow) cluster file systems on top for shared access
- Remote memory
  - L4 network cache – pooled memory
  - Memory allocation functions can be replaced by using LD_PRELOAD.
- Direct access
  - Native I/O library: KDSA-API (Kove Direct Storage Access)
  - Fastest access to XPDs

But none of these access methods provides a shared storage

Our MPI-IO Driver

- The MPI-IO Driver[1]
  - is built on top of the KDSA-API **(Kove Direct Storage Access)**
  - implements many MPI-IO functions
  - is implemented as a shared library and usable with any MPI
  - can be selected at startup of an application using LD_PRELOAD with the shared library
  - checks the file name for the prefix “xpd:”
    - without the prefix, they route the accesses to the underlying MPI
    - files can be selectively stored on XPD volumes
  - There is **no cache** on client side (needed)!

[1] Repository: `http://github.com/JulianKunkel/XPD-MPIIO-driver`

Test-Setup

- Cooley
  - Visualization cluster of Mira on ALCF
  - 3x XPD with 14 FDR connections in total
  - Connected to a GPFS system
- Mistral
  - HPC deployed at DKRZ
  - 3000 compute nodes
  - FDR interconnect
  - Lustre storage system with 54 PByte
    - Peak transfer rate 450 GiB/s

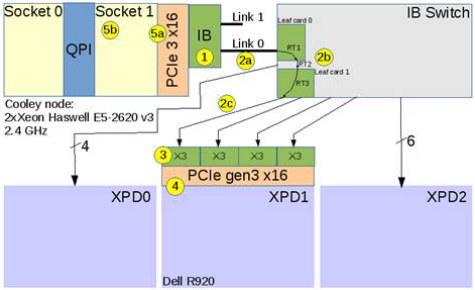
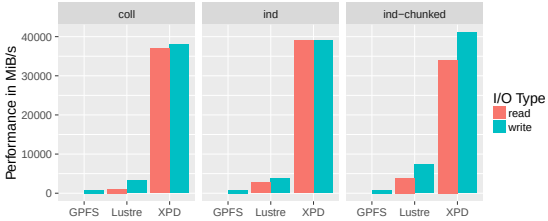


Figure: Cooley XPDs (Kove)

Collective vs. Indepenent vs. Ind.-Chunked (NetCDF-Bench)

Nodes	10
Processes per node	1 (10 if chunked)
Pre-Filling	yes



Observations:
XPDs

- show always best performance
- seem to be insensitive to collective, indendent and independent-chunked I/O

DDN IME240 Specifications

CPU	Dual Intel E5 Series Processor
Memory	8x 16GB, DDR4-2400 RAM
Fabric Ports	2x InfiniBandTM EDR/FDR
Capacity	up to 23x 1.2TB NVMe SSD drives
Data Protection	Erasure coding

Access modes

- IME-native
  - Native I/O library provides the fastest and direct access to IMEs
- IME-FUSE
  - Stores data on IME storage and metadata on Lustre metadata servers
- IME-Lustre
  - A Lustre-backed IME storage
- PMPI-IO with IME support (in development)

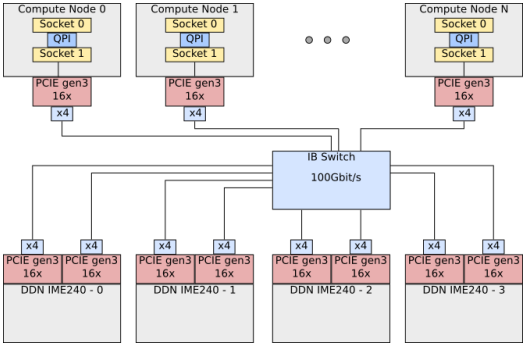
Test cluster

Compute Nodes (1-10x)

CPU	2x Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
RAM	DDR3 8x8GB
Network	1x EDR mlx 5.0 single ported

Storage (4x DDN IME240)

CPU	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
RAM	DDR4 8x16GB
Network	2x EDR mlx 5.0 single ported



Testing NetCDF4 features: Test setup

- Each test configuration is repeated 10 times.
- All experiments are conducted with block sizes 16, 100, 1024, and 10240 KiB.
- **IME-Lustre**, **IME-FUSE** - cache data on IME. The data is synchronized with Lustre storage.

Comparison: IOR-MPIIO vs. NetCDF-Bench on IME-Lustre

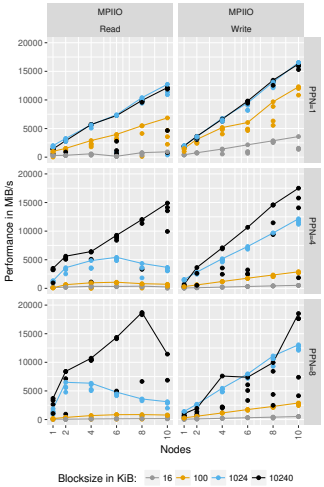


Figure: NetCDF-Bench

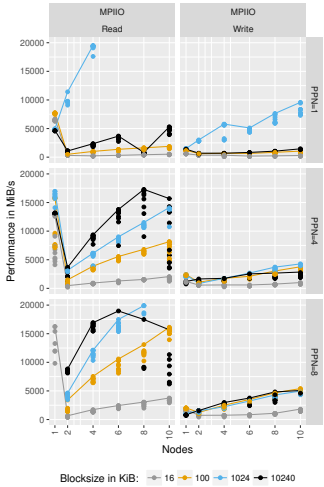


Figure: IOR sequential

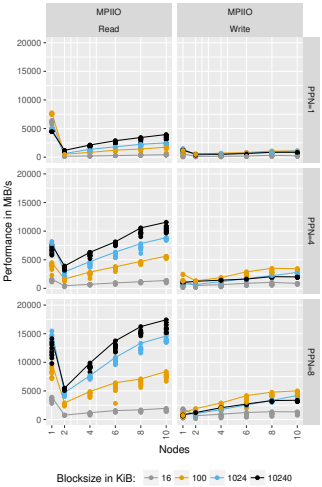


Figure: IOR random

Comparison: IOR-MPIIO vs. NetCDF-Bench on IME-FUSE

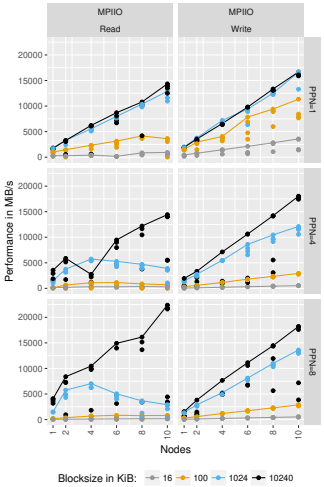


Figure: NetCDF-Bench

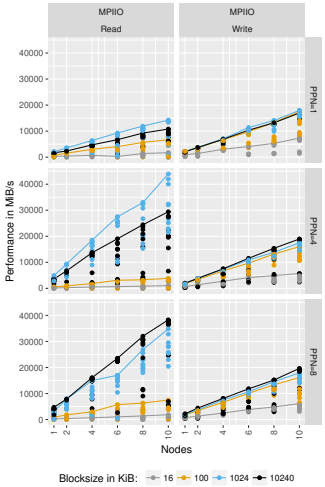


Figure: IOR sequential

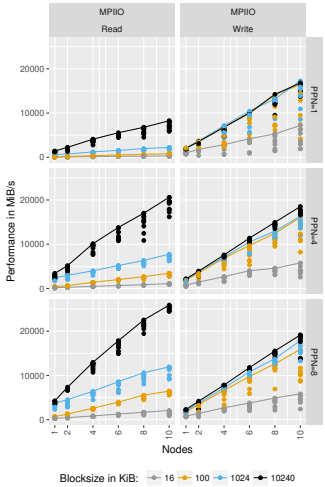


Figure: IOR random

Testing NetCDF4 features with NN=10 and PPN=8 [1]

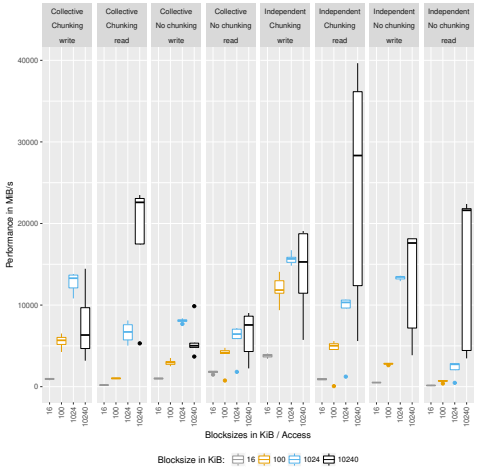


Figure: Linear scale

Observations:

- Large variability for large block sizes
- Independent I/O with chunking works best for large block sizes

Testing NetCDF4 features with NN=10 and PPN=8 [2]

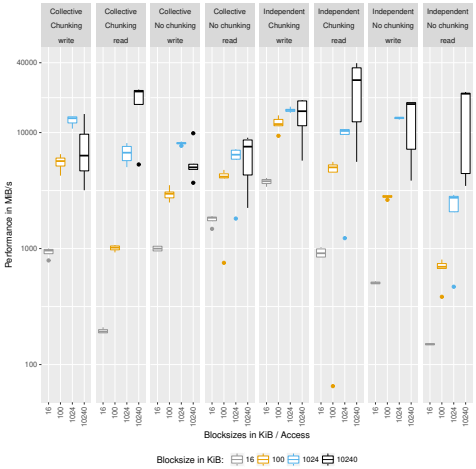


Figure: Logarithmic scale

Observations:

- Collective I/O without chunking or independent I/O with chunking works best for small block sizes

Table Of Content

Scalability (direct access over native interfaces)

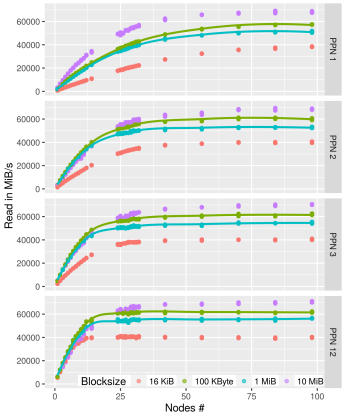


Figure: XPD read

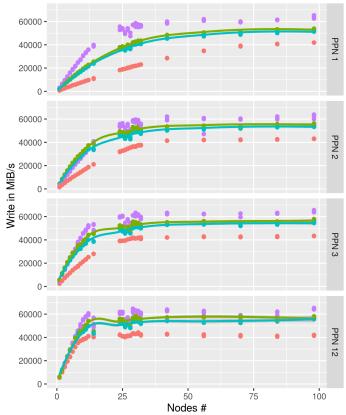


Figure: XPD write

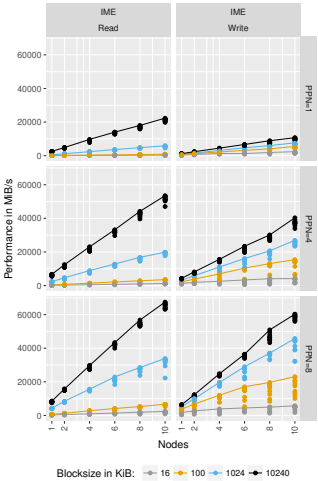


Figure: IME read/write

Summary

- IOR is
  - good for pushing systems to the limits
  - bad for
    - generating access patterns of scientific applications
    - testing special HDF5 features
- NetCDF-Bench mimics data access of scientific applications
- Burst Buffers
  - Kove XPD
    - MPI-IO driver with XPD support makes NetCDF4 performance insensitive to collective/independent I/O and chunking
  - DDN IME
    - Large performance gap between accessing small and large block sizes
    - No information if data is located in IME or on PFS
    - Read performance and variability depends on data location
    - Applications require tuning for better performance