An MPI-IO In-Memory Driver for Non-Volatile Pooled Memory of the Kove® XPD

Julian M. Kunkel, Eugen Betke

German Climate Computing Center

2017-05-22

Group Wissenschaftliches Rechnen (Scientific Computing)

Composed of DKRZ research division and Universität Hamburg research group

Research

- Analysis of parallel I/O
- I/O & energy tracing tools
- Middleware optimization
- Alternative I/O interfaces
- Data reduction techniques
- Cost & energy efficiency

1 Introduction
- In-Memory Storage

2 MPI-IO Driver
- KDSA-API
- MPI-IO Driver

3 Evaluation
- Test Environment
- Results
- 1. Scaling Clients
- 2. Scale-out Performance on 14 Nodes
- 3. Performance Variability: Overview
- 4. Open/Close Times

4 Conclusion

In-Memory Storage

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

Kove XPD Specifications [**xpdspecs**]

Source: Kove XPD L3 datasheet [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

Contribution

Goals

- Provide an MPI-IO implementation for the pooled memory of the XPD
  - Offering shared and parallel file access
- Investigate the performance of the developed MPI-IO driver
  - See if we can saturate the available network bandwidth
  - Evaluate the benefit of the approach for applications using NetCDF/HDF5

Approach

- Use the KDSA-API to implement an PMPI library to support arbitrary MPI

Outline

1 Introduction

2 MPI-IO Driver
- KDSA-API
- MPI-IO Driver

3 Evaluation

4 Conclusion

KDSA-API

- Can access XPD as virtual address space
  - Connect syntax: <local address>/<server>.<link>:<volume ID>
  - Multiple volumes and client or server links can be aggregated
- Send and receive data using write/read calls by utilizing RDMA
  - Synchronous or asynchronous data transfer
- Memory can be pre-registered for use with the Infiniband HCA

MPI-IO Driver I

- The MPI-IO Driver[1]
  - is built on top of the KDSA-API
  - 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
  - checks the file name for the prefix “xpd:”
    - without the prefix, it routes the accesses to the underlying MPI
    - $\Rightarrow$ 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`

MPI-IO Driver II

- Limitations
  - The KDSA calls for unregistered memory are used
    - because memory regions of I/O calls are usually unknown
    - this implies registration overhead to the IB card
  - MPI_Open()/close calls establish/destroy Infiniband connections
    - this implies overhead to these calls
    - but: offers the freedom to choose the volumes/required parallelism on a file basis
  - File views are only partially supported (as needed by NetCDF4/HDF5)

Outline

1 Introduction

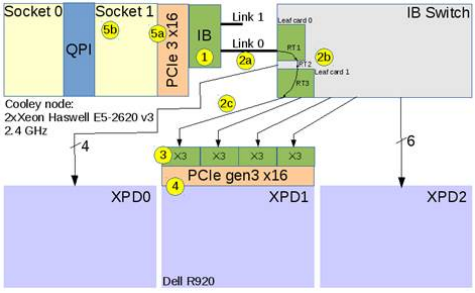
2 MPI-IO Driver

3 Evaluation
- Test Environment
- Results
- 1. Scaling Clients
- 2. Scale-out Performance on 14 Nodes
- 3. Performance Variability: Overview
- 4. Open/Close Times

4 Conclusion

Test Environment

- Cooley
  - ALCF's Mira visualization cluster
  - 126 compute nodes
  - 3x XPDs with 14 FDR connections
  - ⇒ Peak: 70+ GiB/s
  - GPFS storage system
- Mistral
  - DKRZs supercomputer
  - 3000 compute nodes
  - FDR interconnect
  - Lustre storage (54 PByte)
    - Two file systems
    - Peak transfer rate 370 GiB/s



Cooley XPDs [Kove]

Experiments

Benchmarks

- IOR
  - Barriers between the phases are used to synchronize the processes
  - Patterns: shared file, random and sequential
- IO-Model-Test: Measures latencies for 1 thread

Conducted experiments

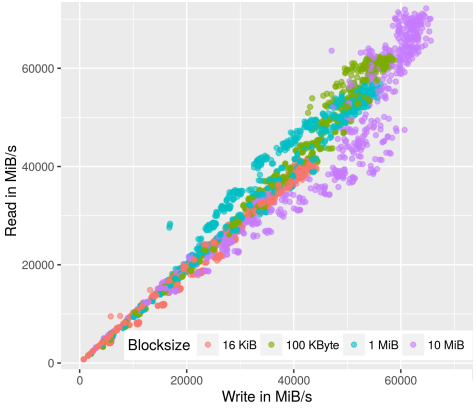
1. Scaling clients with 14 connections each
2. Scale-out performance on 14 nodes with the number of connections
3. Variability of performance
4. Open/close time

For some experiments: a comparision to Lustre (nearly-exclusive) + GPFS

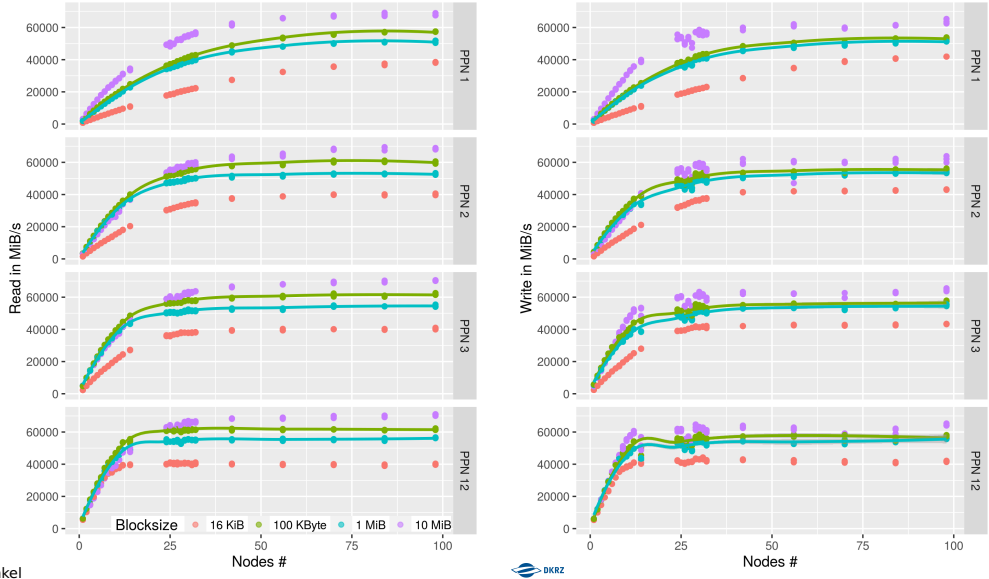
Overview of the Results

- About 7500 different experiments
  - random/sequential I/O
  - read/write I/O
  - different blocksizes, NN, PPN, ...
- Observations
  - Independent from parameters: read $\approx$ write
  - Near network bandwidth
- With open/close() time, 10% lower
  - But files are small $<$ 100 GB

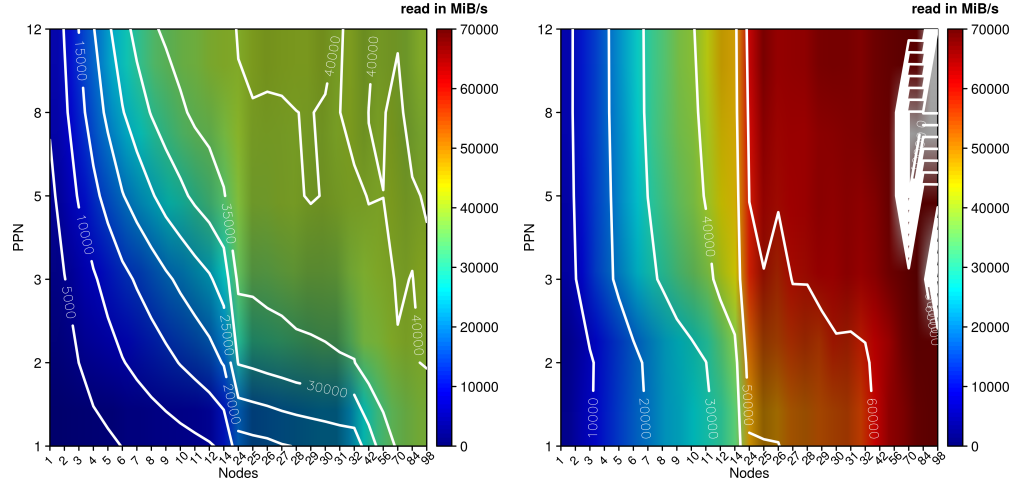
  $\Rightarrow$ We report only actual I/O time



1. Scaling Clients Varying Node Count, PPN, Block Size



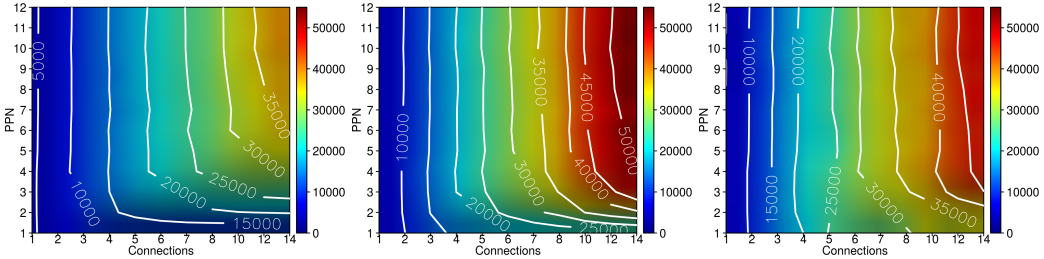
Performance Map for Reads



16 KiB and 1 MiB accesses (beware the color scaling)

2. Scale-out Performance on 14 Nodes

- Constant: 14 compute nodes
- Variable: PPN and number of connections
- Read performance (write similarly)

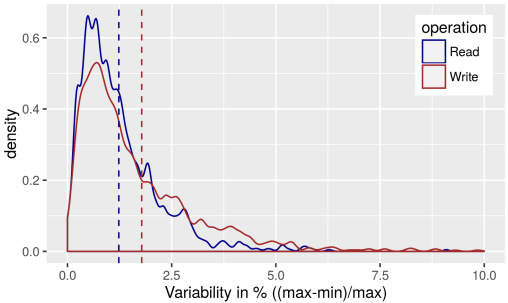


(a) Blocksize: **16 KiB** (b) Blocksize: **100 KByte** (c) Blocksize: **10 MiB**

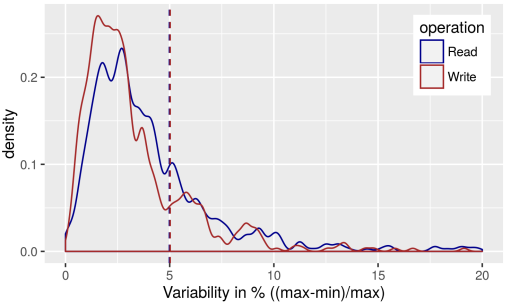
Isolines for multiples of 5000 MiB/s are shown

3. Performance Variability: Overview

- Density of the variability across all conducted experiments
  - Subtracting (max-min) for each configuration
  - Three measurements have been made

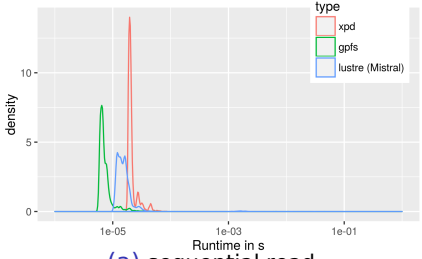


(a) XPD on Cooley

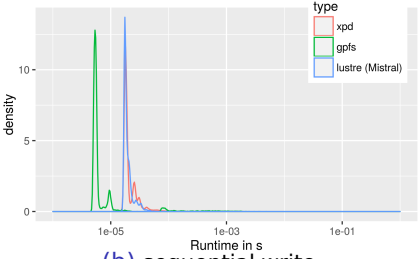


(b) Lustre on Mistral

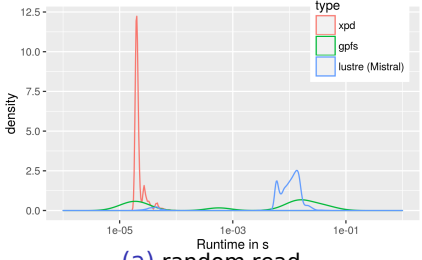
Variability 16 KiB



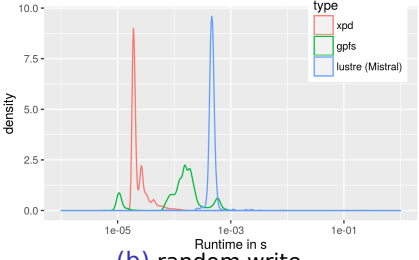
(a) sequential read



(b) sequential write

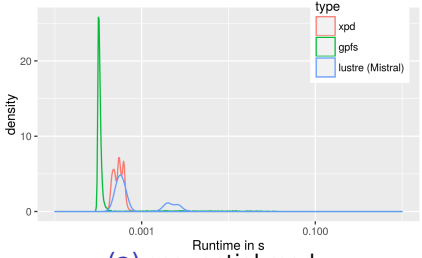


(a) random read

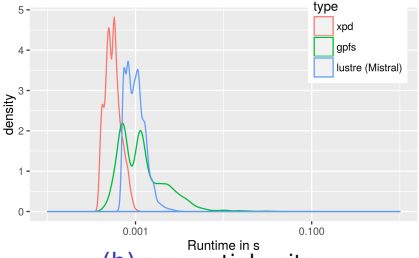


(b) random write

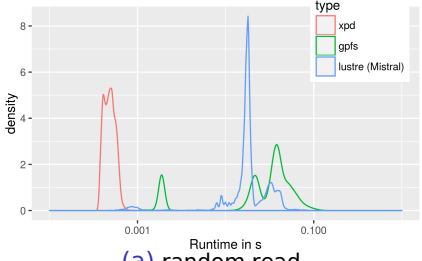
Variability 1 MiB



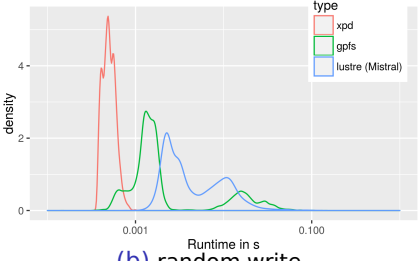
(a) sequential read



(b) sequential write

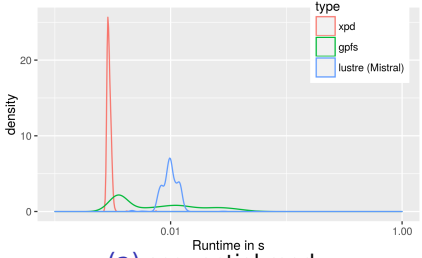


(a) random read

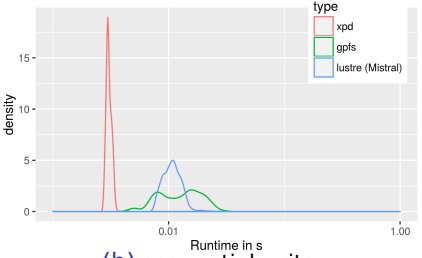


(b) random write

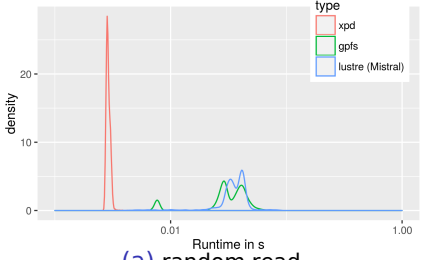
Variability 10 MiB



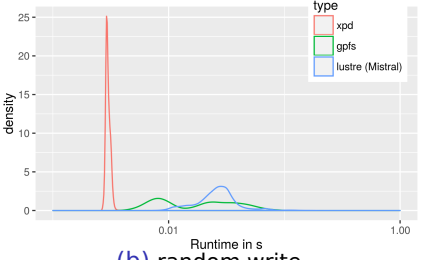
(a) sequential read



(b) sequential write



(a) random read



(b) random write

Mean Performance for the Variability Test

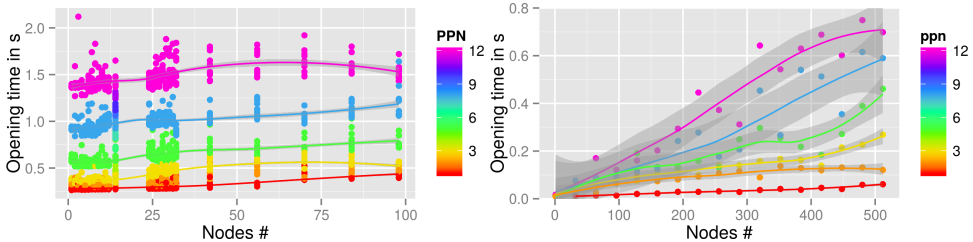
- Table shows the mean harmonic performance (performance of mean time)
- Individual operations of GPFS and Lustre faster sometimes
  - Mean is still worse!
  - Reason: A few very slow operations (barely seen on the figures)

Size	Type	Read			Write		
		XPD	GPFS	Lustre	XPD	GPFS	Lustre
16 KiB	seq	707.8	659.8	522.0	709.8	533.0	778.0
100k	seq	1653.8	1139.2	1082.2	1773.3	611.7	927.7
1 MiB	seq	1837.3	1062.5	996.2	1768.2	629.8	965.9
10 MiB	seq	3401.7	928.3	994.3	3274.3	742.3	916.9
16 KiB	rnd	676.8	1.2	1.5	600.4	71.7	20.6
100k	rnd	1538.5	4.7	9.2	1636.1	346.7	80.6
1 MiB	rnd	2052.6	29.6	49.2	1967.1	184.6	157.6
10 MiB	rnd	3456.6	301.2	277.6	3335.6	430.0	352.1

Variability test: mean performance in MiB/s over the runtime

4. Open/Close Times

- Analyzing the open/close times for all conducted experiments



(a) XPD on Cooley (b) Lustre on Mistral

MPI-IO opening times including fitting curves for PPN $\in \{1,2,3,5,8,12\}$

Summary

- Read performance $\approx$ write performance
- Random I/O $\approx$ sequential I/O
- Scalable performance in terms of client node and number of connections
- For small blocksizes: bottlenecks are CPU and network latency
- Excellent access time variability
- Open/close times reduce mean performance; does not matter for large files
- We also did measurements for NetCDF: also good performance

References


```
1 http://kove.net/downloads/Kove-XPD-L3-datasheet.pdf
```