High-Performance I/O @ Mistral

Julian M. Kunkel

kunkel@dkrz.de

German Climate Computing Center (DKRZ)

06-10-2015

Outline

1 Mistral's Storage System

2 Performance

3 Tunables

4 Obstacles and R&D

ClusterStor Servers

Photo by Carsten Beyer

I/O Architecture (to be extended)

- 29 ClusterStor 9000 Scalable Storage Units (SSUs)
  - SSU: Active/Active failover server pair
- Single Object Storage Server (OSS)
  - 1 FDR uplink
  - GridRaid: (Object Storage Target (OST))
    - 41 HDDs, de-clustered RAID6 with 8+2(+2 spare blocks)
    - 1 SSD for the Log/Journal
  - 6 TByte disks
- 29 Extensions (JBODs)
  - Do not provide network connections
  - Storage by an extension is managed by the connected SSU
- Multiple metadata servers
  - Root MDS + 4 DNE MDS
  - Active/Active failover (DNEs, Root MDS with Mgmt)
  - DNE phase 1: Assign responsible MDS per directory

Parallel File System

Lustre 2.5 (+ Seagate patches: some backports)

Distribution of data

- Lustre can distribute a file across multiple servers (and storage devices called OST)
- Stripe size: Amount of data per OST
- Stripe count: Number of OSTs to store data

General Performance Characteristics

- Client-side caching of reads/writes
  - Dirty data can be drained later
- No server-sided caching
  - I/O requests are directly served by a local file system
- Locking for consistency
  - Read/writes require some communication
- Pre-fetching for sequential reads

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 :-)

Peak Performance

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

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	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

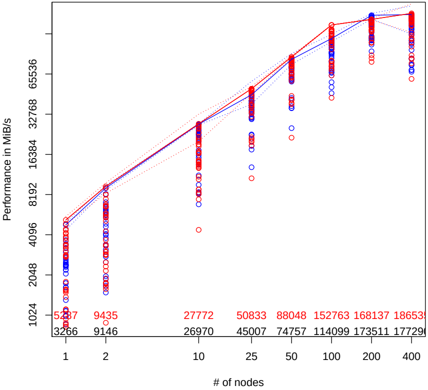
Observations to Take Away

- 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 has shown a high sensitivity with proper configuration
  - 4x improvement is often easy to achieve

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

Performance with Variable Lustre Settings

- Goal: Identify good settings for I/O
- 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



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

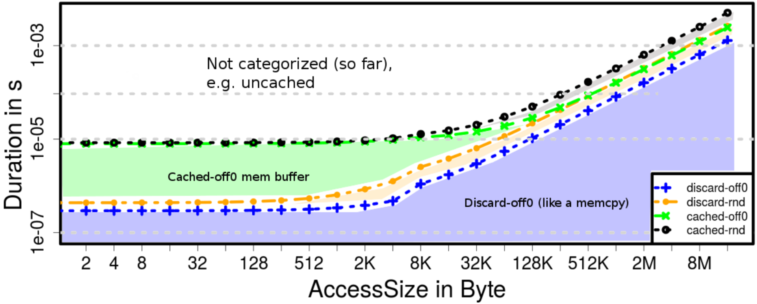
Testing Defaults: Which Stripe Count & PPN to pick?

Average performance values per node (running on various number of nodes)

Stripes	PPN	RNode	WNode	R arithmetic mean	W arithmetic mean	RHarmonic	WHarmonic
1	1	788	780	34	32	52	53
1	2	1214	1155	53	47	75	71
1	4	1352	1518	59	62	81	79
1	6	2179	1835	95	75	91	88
1	8	1943	2235	84	92	86	93
1	12	1974	1931	86	79	92	84
1	16	1890	1953	82	80	84	72
2	1	734	763	32	31	51	51
2	2	1165	1182	50	48	72	72
2	4	1814	1745	79	71	87	85
2	6	1935	1693	84	69	88	83
2	8	1726	2039	75	84	88	89
2	12	1780	2224	77	91	90	92
2	16	1806	1752	79	72	79	75
4	1	726	761	31	31	49	51
4	2	1237	1185	54	48	70	66
4	4	1737	1744	75	71	85	84
4	6	1719	1888	75	77	85	86
4	8	1751	1931	76	79	87	90
4	12	1841	1972	80	81	87	89
4	16	1745	2064	76	85	72	74
16	1	743	726	32	29	48	49
16	2	1109	1216	48	50	66	71
16	4	1412	1554	61	64	75	81
16	6	1489	1812	65	74	72	85
16	8	1564	1841	68	75	79	90
16	12	1597	1939	69	79	71	78
16	16	1626	1900	71	78	64	68
116	1	588	432	25	17	34	31
116	2	871	773	38	31	44	52
116	4	1270	1258	55	51	53	69
116	6	1352	978	59	40	52	51
116	8	1397	901	61	37	56	47
116	12	1470	1020	64	42	55	46
116	16	1503	1147	65	47	55	42

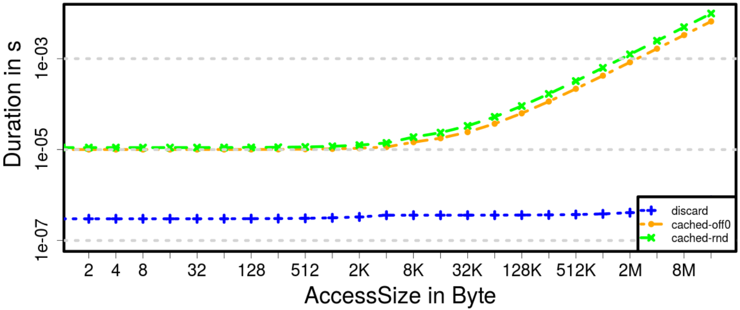
I/O Duration with Variable Block Granularity

- 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

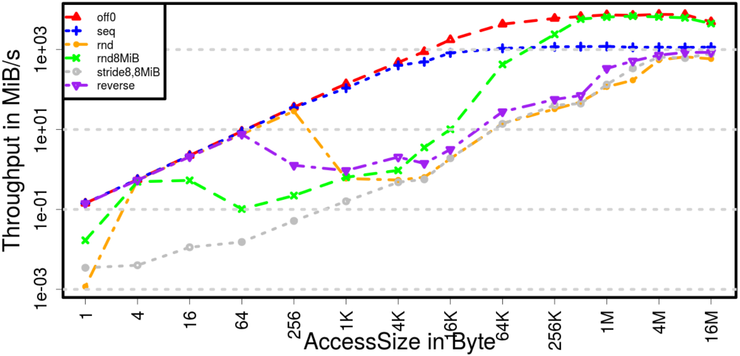


Write

- Memory layout has a minor impact on performance

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

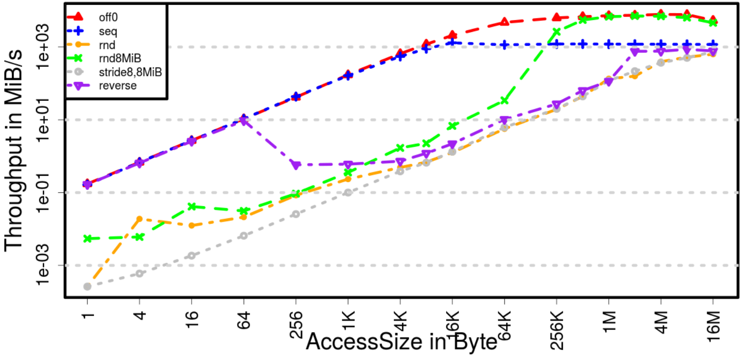
Throughput with Variable Granularity



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

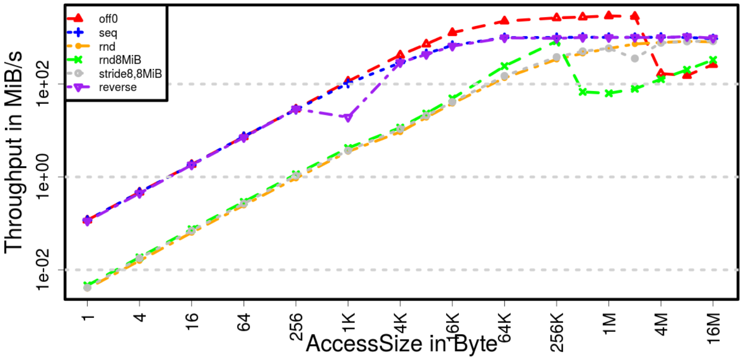
Throughput with Variable Granularity



Read – clean cache

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

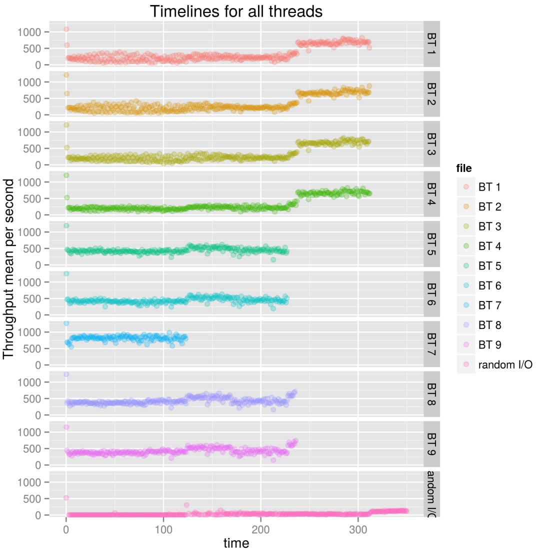
Throughput with Variable Granularity



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



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 intense 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
```

Obstacles We Face When Are Optimizing the System

Lack of knowledge

- Usage of file formats and middleware libraries is limited
  - Analysis of file extensions does not suffice
  - Library usage could theoretically be monitored, but ...
- The workflows are sometimes diffuse
- The cause of innefficient operations is unknown

Shared nature of storage

- With 1/60th of nodes one can drain 1/7th of I/O performance
  - $\Rightarrow$ 10% of nodes drain all performance
  - Since applications are not doing I/O all the time this seems fine
- But: interaction of I/O may degrade performance
  - I/O intense benchmark increased application runtime by 100%
- Metadata workloads are worse, problematic with broken scripts

Obstacles

Difficulties in the analysis

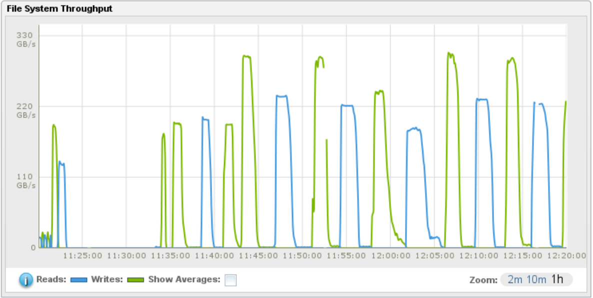
- Performance is sensitive to I/O patterns, concurrent activitity
- Infiniband oversubscription
- Application-specific I/O servers increase complexity
- Capturing a run's actual I/O costs vs. shared access
- Lustre's (performance) behavior

Others

- Outdated (and inefficient) file formats are still dominant
- Capability increase from Blizzard to Mistral[3]
  - Compute performance by 20x
  - Storage performance by 20x
  - Storage capacity by 7x

[3] This is a projection for the full system

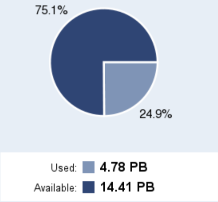
Monitoring I/O Performance with ClusterStor



Top System Statistics

	Metric		
File System	Peak Read	274.34	GB/s
	Current Read	45.6	KB/s
	Peak Write	308.37	GB/s
	Current Write	222.54	GB/s
Metadata	Current Operations	0	Op/s
Storage	Number of OSTs in use	116	
	Number of Disks in use	4,882	
	Capacity in use	4.78	PB
	Capacity available	14.41	PB
Power	Current cluster usage	70.17	KW

Capacity Overview



Lustre I/O Statistics

- Statistics on the client help understand behavior (a bit)
- `/proc/fs/lustre/llite/lustre01-*/stats`
- `/proc/fs/lustre/llite/lustre01-*/read_ahead_stats`

Typ	Lay-out	Acc-Size	numa_local	hits	misses	intr	softirq	read b_avg	read calls	write b_avg	write calls	osc_read avg	osc_read calls	osc_write avg	osc_write calls	Perf. in MiB/s
W D	off0	256K	263K	0	0	0.9-1K	1.8-2K	201	3	40K	5	0	0	32K	0-6	1.1T
W C	off0	256K	264K	0	0	2.8-3.3K	6.1-7.1K	201	3	262K	10005	0	0	256K	1.1	2.6G
W C	seq	256K	940K	0	0	16-18K	26-30K	201	3	262K	10005	0	0	4M	625	1G
W C	rnd	256K	937K	0	0	125K	34K	201	3	262K	10005	4096	19K	3.9M	673.6	341M
W C	rev	256K	942K	0	0	23K	28-77K	201	3	262K	10005	0	0	4M	626	963M
R D	off0	256K	263K	0	0	1.1-1.4K	2.4-3K	201	3	40K	5	0	0	42K	0.4	14G
R C	off0	256K	264K	63	1	1.4-1.9K	2.9-3.9k	256K	10003	40K	5	256K	1	0	0	5.9G
R C	seq	256K	931K	640K	3	25-60k	28-111K	256K	10003	57K	5	1M	2543	80K	0.4	1.1G
R C	rnd	256K	1559K	615K	16K	136-142k	43k-65k	256K	10003	58K	5	241K	20K	180K	4	33M
R C	rev	256K	930K	629K	10K	70-77K	23-47K	256K	10003	58K	5	256K	9976	104K	0-3	56M
R U	off0	256K	264K	63	5	1.5-2k	2.9-3.9k	256K	10003	40K	5	64K	5	0	0	6.2G
R U	seq	256K	946K	640K	6	25-42k	32-74k	256K	10003	57K	5	1M	2546	0	0	1.2G
Runs with accessSize of 1 MiB and a 1 TB file, caching on the client is not possible. For seq. 1M repeats are performed, for random 10k:																
W	seq	1M	259M	0	1.3	8-12M	14-23M	201	3	1M	1000013	0-8K	0-4	4M	250K	1007
W	rnd	1M	2.9M	0	0-3	161K	114K	201	3	1M	10006	4097	20K	3.2M	3309	104
R	seq	1M	257M	255M	2	16-22M	28-38M	1M	1000003	2.5M	12	1M	1000K	3M	10	1109
R	rnd	1M	5M	2M	9753	206K	157-161K	1M	10003	60K	5	836K	24K	100K	3	55
Accessing 1 TB file with 20 threads, aggregated statistics, but performance is reported per thread:																
W	seq	1M	260M	0-1	0-3	12M	23M	201	58	1M	990K	2-17K	1-3	4.1M	254K	250
W	rnd	1M	246M	0	0	18M	13M	201	58	1M	960K	4096	1.8M	3.1M	320K	138
R	seq	1M	254M	250M	480K	9.8M	12M	1M	970K	21-24K	0.2-1.2K	1.6M	630K	717K	41	168
R	rnd	1M	481M	240M	900K	20M	16M	1M	950K	20-23K	0.2-1.2K	832K	2.3M	523K	36	47

Deltas of the statistics from /proc for runs with access granularity of 256 KiB and 1 MiB (mem-layout is always off0). In the type column, D stands for discard, C for cached and U for uncached. 1TB files do not fit into the page cache.

Relevant R&D at DKRZ

We are doing various R&D to improve the situation:

- Monitoring and analysis of I/O on system AND application level
- Optimized data layouts for HDF/NetCDF
- QoS for I/O (interactive vs. large scale runs)
- Evaluate of alternative storage for random workloads
- Compression of data (lossless 1:2.5, lossy $>$ 1:10)
- ! At best without changes on YOUR applications

Please talk to us, if you think you have an I/O issue

Appendix

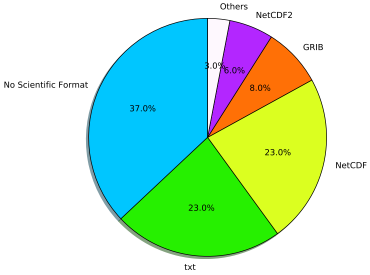
File Formats

- Problem: File extensions do not match the content

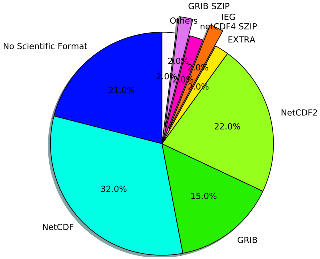
⇒ Sample of files analyzed with `file` and cdo

- 25% from home
- 20% from work/scratch: 1 PB, 26 M files

Scientific file formats for work/scratch



% file count



% file size

Insights from File Analysis

Home:

- Not much insight
- Mostly code/objects
- Many empty directories, broken links ...

Work/Scratch:

- Many old/inefficient file formats around
- Many small files + TXT
- A small fraction of data volume is compressed:
  - 2% NetCDF and 2% GRIB SZIP, 3% GZIP compressed
- A small fraction (3% of volume) of NetCDF4/HDF5

Dealing with Storage in ESiWACE

H2020 project: ESiWACE Center of Excellence

Work package 4

Partners: DKRZ, STFC, ECMWF, CMCC, Seagate

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