Lustre and IO-500

Experiences with the Cambridge Data Accelerator

Matt Rásó-Barnett
University of Cambridge
LAD'19

UNIVERSITY OF CAMBRIDGE

Overview

- ➔ Obtained #1 position in June ISC'19 IO-500 Full-List with our all-flash Lustre-based burst-buffer
- ➔ This was our second submission on the same hardware; overall score improved nearly 4x over November 2018 List
- ➔ This improvement came nearly entirely from software changes
  - Used the very latest features Lustre has to offer
    - LNET multirail
    - DNE2 - 24x MDTs
    - DoM
    - OST Over-Striping
  - Support and development effort from Whamcloud turbo-charged our effort, almost doubling our previous best score of the time
  - Tuning for benchmark
- ➔ Provides an interesting view on what's possible with the latest Lustre releases

IO-500 Overview

Before:

#	information							io500		
	institution	system	storage vendor	filesystem type	client nodes	client total procs	data	score	bw	md
									GiB/s	kIOP/s
2	University of Cambridge	Data Accelerator	Dell EMC	Lustre	528	4224	zip	158.71	71.40	352.75

After:

1	University of Cambridge	Data Accelerator	Dell EMC	Lustre	512	8192	zip	620.69	162.05	2377.44

Headline 'SCORE' in the IO-500 is combination of individual metrics, primarily based on Bandwidth and Metadata performance.

For our score, Metadata performance was the dominant contribution to improvement

IO-500 Overview

Talk at LUG'19: *IO-500 – A Storage Benchmark for HPC – Andreas Dilger, Whamcloud*

Made up of 5 Benchmark Scenarios

- Designed to provide balanced overall measurement
- Combination of 'best' and 'worst' case IO patterns

IOR 'easy'	Write and Read	Free to tune IOR parameters. Typically file-per-process, large, aligned chunks
IOR 'hard'	Write and Read	Limited options. Forced to use small, unaligned IO to a single shared file
mdtest 'easy'	Create, Stat, Delete	Free to tune mdtest parameters. Separate directory per process. Zero-size files.
mdtest 'hard'	Create, Stat, Delete	Limited options. Forces all process writing to single shared directory. 3901 byte files.
find	Namespace search	Find specific subset of files from those created in other 4 benchmarks. `-newer {timestamp} -size {mdtest_hard_size} -name *01*`

IO-500 Overview

BW	phase 1	ior_easy_write
IOPS	phase 1	mdtest_easy_write
BW	phase 2	ior_hard_write
IOPS	phase 2	mdtest_hard_write
IOPS	phase 3	find
BW	phase 3	ior_easy_read
IOPS	phase 4	mdtest_easy_stat
BW	phase 4	ior_hard_read
IOPS	phase 5	mdtest_hard_stat
IOPS	phase 6	mdtest_easy_delete
IOPS	phase 7	mdtest_hard_read
IOPS	phase 8	mdtest_hard_delete

12 Test Phases overall

- 4 Bandwidth Phases (ior_easy_write, ior_easy_read...)
- 8 IOPS Phases (mdtest_easy_write, _stat, _delete, ...)

Each phase has a result - either GB/s for bandwidth or kIOPs for metadata

All **WRITE** phases must run for 300s minimum to be valid submission

$$\sqrt[4]{(ior\_easy\_read) * (ior\_easy\_write) * (ior\_hard\_write) * (ior\_hard\_read)} = Bandwidth\ Score$$

$$\sqrt[8]{(mdtest\_easy\_write) * (mdtest\_easy\_stat) * (mdtest\_easy\_delete) * \ldots} = Metadata\ Score$$

Geometric Mean of both

=

Total Score

IO-500 - How to get started - (Pt. 1)

```
$ git clone https://github.com/VI4IO/io-500-dev
$ cd io-500-dev

# Load your cluster's preferred MPI library
$ module load intelmpi...

# Run compilation script
$ ./utilities/prepare.sh
…
OK: All required software packages are now prepared
io500_fixed.sh  ior  mdtest  mmfind.sh  pfind  sfind.sh  testlib

$ ls
bin  build  CHANGELOG.md  doc  io500.sh  lib  README.md  site-configs  utilities
```

The `io500.sh` script is what you will use to run the benchmark.
Simply run this in your batch job script

IO-500 - How to get started - (Pt. 2)

`io500.sh` script contains some simple functions that can be edited before running:

```
function setup_directories {
 # set directories for where the benchmark files are created and where the results will go.
 io500_workdir=/lustre/scratch/io500                # directory where the data will be stored
 io500_result_dir=$PWD/results/$timestamp           # the directory where the output results will be kept
 timestamp=`date +%Y.%m.%d-%H.%M.%S`

 io500_ior_easy=$io500_workdir/ior_easy
 io500_ior_hard=$io500_workdir/ior_hard
 io500_mdt_easy=$io500_workdir/mdt_easy
 io500_mdt_hard=$io500_workdir/mdt_hard

 mkdir -p $io500_workdir $io500_result_dir $io500_ior_easy $io500_ior_hard $io500_mdt_easy $io500_mdt_hard

 # set striping settings on directories here
 ...
}

function setup_paths {
 # Set the paths to the binaries.  If you ran ./utilities/prepare.sh successfully, then binaries are in ./bin/
 io500_ior_cmd=$IO500_DIR/bin/ior
 io500_mdtest_cmd=$IO500_DIR/bin/mdtest
 io500_mdreal_cmd=$IO500_DIR/bin/md-real-io
 io500_mpirun="srun"
 io500_mpiargs="--mpi=pmi2 --cpu_bind=socket"
}
```

IO-500 - How to get started - (Pt. 3)

OUT OF DATE

Script also contains permitted parameters to tweak for each benchmark:

```
function setup ior easy {
 io500 ior easy size=160
 # 16M transfer size, 160GB per proc, file per proc, O DIRECT
 io500_ior_easy_params="-a=POSIX --posix.odirect -C -t 16m -b ${io500_ior_easy_size}g -F"
}

function setup mdt easy {
 io500 mdtest easy params="-u -L" # unique dir per thread, files only at leaves
 io500_mdtest_easy_files_per_proc=540000
}

function setup ior hard {
 io500 ior hard writes per proc=90000
 io500_ior_hard_other_options="-a MPIIO" #e.g., -E to keep precreated files using lfs setstripe, or -a MPIIO
}

function setup mdt hard {
 io500 mdtest hard files per proc=32000
 io500_mdtest_hard_other_options=""
}
```

Tweaking these values is where you ~~will~~ ***used to*** spend most of your time!

Current io500.sh simplifies this, removing the need to set number of writes-per-process for example

IO-500 - General Tips

- Get Organised once you are up and running with benchmark
  - Make a separate working directory and parameterize your `io500.sh`
  - Think about how you will keep track of parameters used and results for that configuration

- Repeat configurations!
  - Can be surprising the amount of variation in performance between identical runs
  - Check server/client logs, fabric monitoring - might encounter issues to follow up

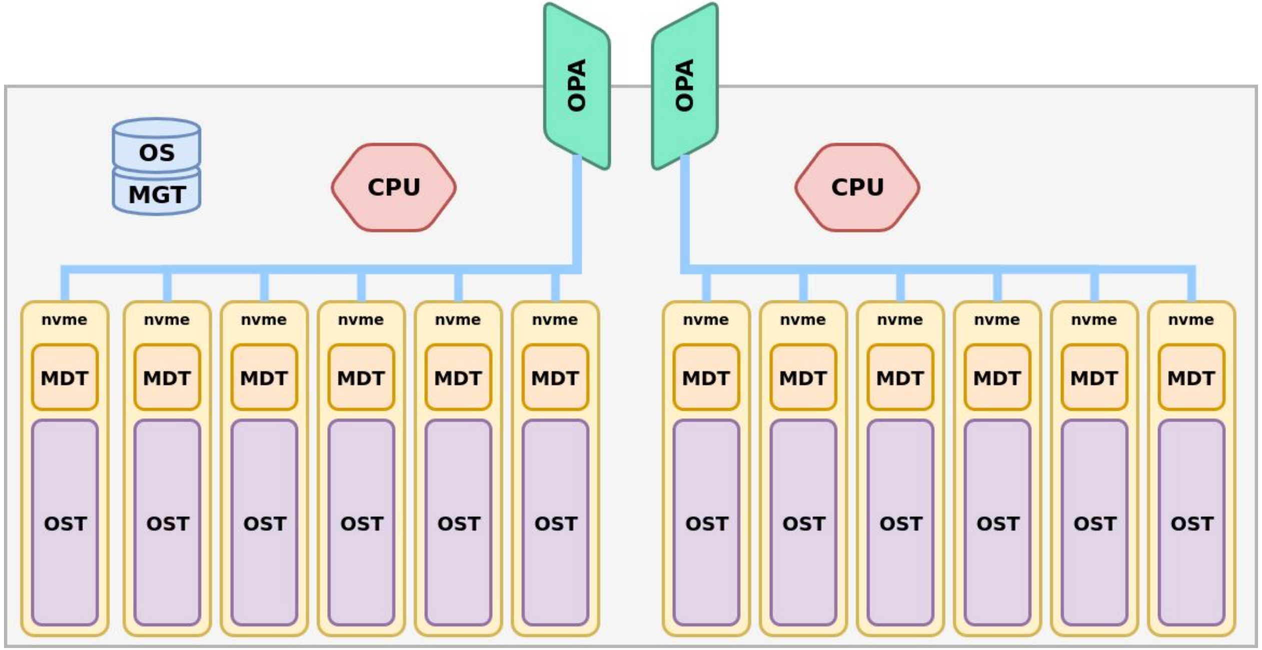
- Time
  - Full run of the benchmark take over an hour
  - Can use variables in io500.sh to evaluate each benchmark stage separately
    eg:
    Run ‘ior_easy’ phases only - probe what works/doesn’t in a quicker feedback loop.
    Helps to find parameter values quickly.

‘Data Accelerator’

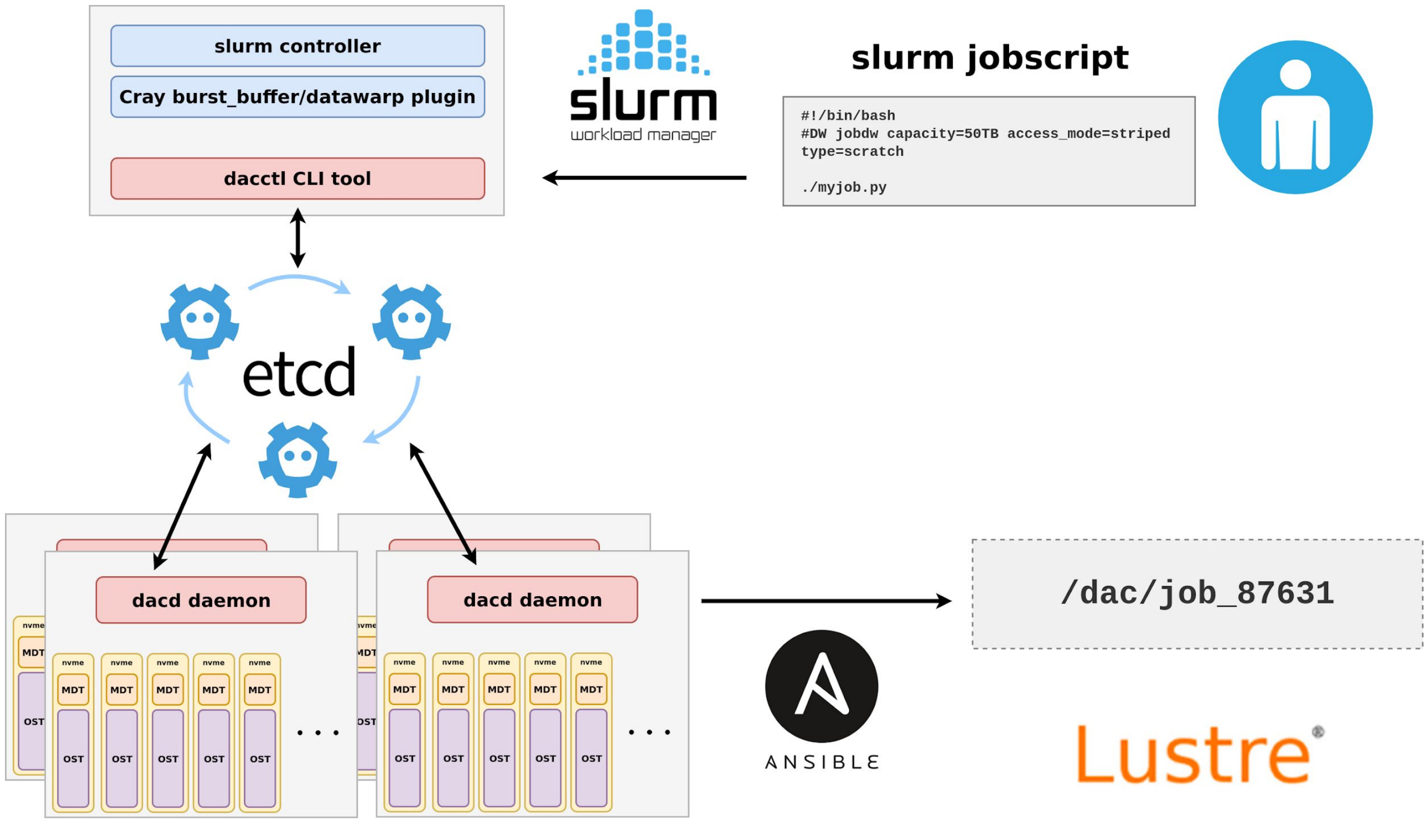
Our all-flash Lustre burst-buffer

Hardware Platform

- 24x Dell R740xd Servers
  - 12x Intel SSD P4600 1.4TiB NVMe per server
  - 2x Intel Omnipath HFIs @100Gbps per server
  - 2x Intel Xeon Gold 6142 CPU 32C @2.60GHz
  - 192GiB DDR4



How we use it - Lustre Filesystems-on-demand



Slurm Integration

https://github.com/RSE-Cambridge/data-acc

- Repo contains installation instructions, as well as quickstart demo environments deployable with Docker or Openstack
- Core code written in Golang, along with Ansible to do Lustre filesystem creation/deletion
- Contributions/Feedback welcome!

DAC in IO-500

- Filesystem-on-demand is perfect for benchmarking.
  - Create/Destroy filesystem with each job, no user-contention, no degradation

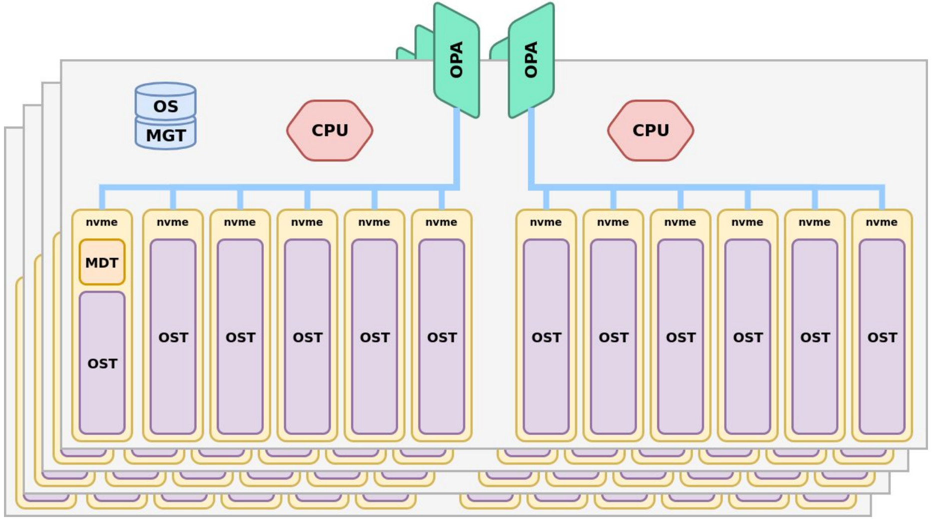
- Allows agility in testing new Lustre versions - no persistency to worry about

- We chose not to utilise any RAID on top of the NVMe devices due to ephemeral nature of any one Filesystem
  - If a device is lost, just restart the job
  - Optimising for performance over long-term resiliency

- All of this helps the system in this kind of benchmark achieve a peak-performance
  - Not really representative of a typical persistent scratch filesystem

Filesystem Layout for IO-500

- Filesystem configuration used all NVMe's
  - 1x MDT per server = **24 MDTs**
  - 12x OSTs per server = **288 OSTs**
- One device per-server partitioned to give MDT and OST on same device.
  - Each MDT was 100GiB in size
  - Each OST approx 1.4TiB
- Overall filesystem:
  ~450TiB in size
  ~1Billion inodes - *Important given number of files created during mdtest_easy at peak performance!*



Benchmarking Process

- Worked with ***customised*** DAC ansible directly
  - Manually overriding the Filesystem layout - the produced Ansible inventory
  - Added new plays to apply Server/Client ‘tunings’

```
  - name: Set client 16M RPCs
    command: "sudo lctl set_param osc.{{ fs_name
}}-OST*.max_pages_per_rpc=16M"
    register: client_max_pages_per_rpc_result
    retries: 3
    delay: 10
    until: client_max_pages_per_rpc_result.rc == 0
```

- Most of this is currently still ***outside*** the DAC upstream code-base
  - Aiming to integrate most of this work in the upstream project for November IO-500 runs
  - Run benchmark on upstream DAC burst-buffer
  - Can then match benchmark score to specific Ansible inventory and DAC codebase version as part of a **reproducible configuration**

```
# Example of produced Ansible Inventory
dac-prod:
 children:
   fs1:
     hosts:
       dac-e-1.fabric.cluster :
         fs1 mdts:
           nvme7n1: 0
         fs1 mgs: sdb
         fs1 osts:
           nvme0n1: 0
           nvme10n1: 9
           nvme11n1: 11
           nvme1n1: 2
           nvme2n1: 4
           ...
       dac-e-10.fabric.cluster :
         ...

     vars:
       lustre checksums : 0
       lustre fs name : fs1
       lustre ldlm lru size : 4000000
       lustre lnet network : o2ib1
       lustre max dirty mb : 512
       lustre max mod rpcs in flight : 127
       lustre max read ahead mb : 2048
       lustre max read ahead per file mb : 256
       lustre_max_rpcs_in_flight : 128
```

Initial Experimentation - April 2019

- Began playing with IO-500 in April around LUG'19
- ***LOTS*** of early failures - trial and error - started out small-scale with few clients.
  - Worked up gradually to larger numbers of clients (10, 32, 64, 128...)
    - Large parameter space to explore. Needed to find right set of io500.sh parameters to produce valid runs at each stage
  - Many problems along the way:
    - Remember to clean up failed runs - can easily fill filesystem with mdtest files
    - Bug in IO-500 pfind, not matching any files (Resolved in https://github.com/VI4IO/io-500-dev/issues/37)
    - Was not using a Slurm reservation of nodes originally, different compute nodes in each run
      - Switched to reservation and team-members did node health-check on compute nodes with HPL which identified some sub-par nodes - got good working set of nodes - important at high client counts
    - Higher node-count jobs in-general was a learning curve - *got help from other team-members about configuring MPI library for our fabric*

Initial Best Large-Scale Results

November 2018
Servers: Lustre 2.11
528 Clients - 8ppn: Lustre 2.10.5

BW	phase 1	ior_easy_write	208.252 GB/s
IOPS	phase 1	mdtest_easy_write	53.451 kiops
BW	phase 2	ior_hard_write	7.441 GB/s
IOPS	phase 2	mdtest_hard_write	366.946 kiops
IOPS	phase 3	find	729.390 kiops
BW	phase 3	ior_easy_read	358.561 GB/s
IOPS	phase 4	mdtest_easy_stat	247.400 kiops
BW	phase 4	ior_hard_read	46.780 GB/s
IOPS	phase 5	mdtest_hard_stat	2112.230 kiops
IOPS	phase 6	mdtest_easy_delete	50.864 kiops
IOPS	phase 7	mdtest_hard_read	1618.130 kiops
IOPS	phase 8	mdtest_hard_delete	389.670 kiops

May 2019
Servers: Lustre **2.12.2**
502 Clients - **16ppn**: Lustre **2.10.7**

ior_easy_write	**337.891 GB/s**
mdtest_easy_write	**1846.370 kiops**
ior_hard_write	**14.413 GB/s**
mdtest_hard_write	**558.397 kiops**
find	**1863.750 kiops**
ior_easy_read	**532.529 GB/s**
mdtest_easy_stat	**3138.370 kiops**
ior_hard_read	**81.216 GB/s**
mdtest_hard_stat	**1053.720 kiops**
mdtest_easy_delete	**1015.380 kiops**
mdtest_hard_read	**772.956 kiops**
mdtest_hard_delete	**441.478 kiops**

Nov 2018: [SCORE] Bandwidth 71.4032 GB/s : IOPS 352.754 kiops : TOTAL 158.707

May 2019: [SCORE] Bandwidth **120.47 GB/s** : IOPS **1103.69 kiops** : TOTAL **364.639**

Why the Improvement?

- Lots of improvements across the board
  - More time dedicated to benchmarking. Some configuration mistakes made when rushing in November 2018
  - Verified ‘known-good’ group of compute nodes → Lessons learned feeding back into general service
  - Identified congested ISLs in Fabric → Reorganised DAC servers to reduce this

  - Lustre **2.12.2** on servers
    - **MANY** performance improvements, particularly for Flash - See talks from LUG2019
      (*Lustre Optimizations and Improvements for Flash Storage – Shuichi Ihara, Whamcloud*)

  - 24x MDTs - Started using DNE at larger scale

    DNE1 remote directories for mdtest_easy: `lfs setdirstripe -c -1 $mdtest_easy_testdir`
    DNE2 striped directory for mdtest_hard: `lfs setdirstripe -c -1 -D $io500_mdt_hard`

  - More client processes-per-node: 8ppn →16ppn

  - POSIX O_DIRECT on ior_easy. MPIIO on ior_hard
    Servers/Clients tuned to 16MiB RPCs.

    - Managed to get much closer to ‘peak’ ior_easy bandwidth

Final Large-Scale Submission

May 2018
Servers: Lustre 2.12.2
502 Clients - 16ppn: Lustre 2.10.7

BW	phase 1	ior_easy_write	337.891	GB/s
IOPS	phase 1	mdtest_easy_write	1846.370	kiops
BW	phase 2	ior_hard_write	14.413	GB/s
IOPS	phase 2	mdtest_hard_write	558.397	kiops
IOPS	phase 3	find	1863.750	kiops
BW	phase 3	ior_easy_read	532.529	GB/s
IOPS	phase 4	mdtest_easy_stat	3138.370	kiops
BW	phase 4	ior_hard_read	81.216	GB/s
IOPS	phase 5	mdtest_hard_stat	1053.720	kiops
IOPS	phase 6	mdtest_easy_delete	1015.380	kiops
IOPS	phase 7	mdtest_hard_read	772.956	kiops
IOPS	phase 8	mdtest_hard_delete	441.478	kiops

June 2019
Servers: Lustre '**master**' branch + patches
512 Clients - 16ppn: Lustre '**master**' branch + patches

ior_easy_write	328.875	GB/s
mdtest_easy_write	1784.640	kiops
ior_hard_write	**50.664**	**GB/s**
mdtest_hard_write	558.621	kiops
find	1721.020	kiops
ior_easy_read	509.259	GB/s
mdtest_easy_stat	**183233.000**	**kiops**
ior_hard_read	81.267	GB/s
mdtest_hard_stat	**5763.560**	**kiops**
mdtest_easy_delete	**1122.060**	**kiops**
mdtest_hard_read	**858.445**	**kiops**
mdtest_hard_delete	**584.785**	**kiops**

May 2019: [SCORE] Bandwidth 120.47 GB/s : IOPS 1103.69 kiops : TOTAL 364.639

June 2019: [SCORE] Bandwidth **162.049 GB/s** : IOPS **2377.44 kiops** : TOTAL **620.695**

Why the Improvement? (Pt.1)

- Engagement from Whamcloud post-LUG'19
  - Discussion about developments going on at the time with specific performance improvements particularly around metadata.
  - Shared with me a number of patches that were landing at the time and tunings that they were testing with
  - Working off 'master' branch - Upgraded all Servers ****and Clients**** to the current tip of 'master' at the time, applied patches on top. **Having 2.12+ clients opened up new features to utilise!**
  - 'master' branch also contained the new Lustre Overstriping feature presented at LUG2019 (*Lustre Overstriping -Improving Shared File Write Performance – Patrick Farrell, Whamcloud*)

```
BW   phase 2        ior_hard_write     14.413 GB/s
BW   phase 4         ior_hard_read     81.216 GB/s
```

→

```
ior_hard_write                  50.664 GB/s
ior_hard_read                   81.267 GB/s
```

```
# Use Lustre OST overstriping - 5x stripes per OST
lfs setstripe -C $((288*5)) -S 16M $io500_ior_hard
```

- Huge ~4x improvement in ior_hard_write performance

Why the Improvement? (Pt.2)

- Data-on-Metadata
  - 2.12+ Clients enabled us to make use of DoM
  - With all NVMe MDTs and OSTs possibly little benefit to create performance (mdtest_easy_write)
  - However combination of DoM + specific patches [LU-12325] [LU-11623] + tunings shared by WC:

```
[MDS]     $ lctl set_param mdt.*.dom_lock=trylock
 # Very Large number of client-side locks in LRU cache
# Function of available server RAM - fortunately we have 24x 192GB MDS
[Clients] $ lctl set_param ldlm.namespaces.*.lru_size=4000000
```

Lead to ****HUGE**** 'mdtest_stat_*' improvements - **particularly** 'mdtest_easy_stat'

```
IOPS phase 4     mdtest_easy_stat   3138.370 kiops
IOPS phase 5     mdtest_hard_stat   1053.720 kiops
IOPS phase 6   mdtest_easy_delete   1015.380 kiops
IOPS phase 7     mdtest_hard_read    772.956 kiops
IOPS phase 8   mdtest_hard_delete    441.478 kiops
```

→

```
mdtest_easy_stat          183233.000 kiops
mdtest_hard_stat            5763.560 kiops
mdtest_easy_delete          1122.060 kiops
mdtest_hard_read             858.445 kiops
mdtest_hard_delete           584.785 kiops
```

mdtest_easy_stat

- Score was suspiciously large. How is 183M stats per second even possible?
  - Essentially for mdtest_easy with DoM, the patches and tunings applied, and the large client 'lru_size' - the files the client has to stat is in cache locally on the node - no need to contact the MDT because it already holds the lock.
  - Possible because mdtest was not running in 'strided' mode - eg: perform 'stat' lookup of a file from different client to the one that created it
  - So this is measuring a cache effect
- This has been noted and fixed in the current IO-500 script
  - mdtest now force the '-N 1' flag to enforce strided behaviour on both 'easy' and 'hard' tests
  - I haven't re-run yet since these changes were introduced - but expecting big reductions in mdtest_stat_* scores

ISC'19 Results

#	information							io500		
	institution	system	storage vendor	filesystem type	client nodes	client total procs	data	score	bw GiB/s	md kIOP/s
1	University of Cambridge	Data Accelerator	Dell EMC	Lustre	512	8192	zip	620.69	162.05	2377.44
2	Oak Ridge National Laboratory	Summit	IBM	Spectrum Scale	504	1008	zip	330.56	88.20	1238.93
3	JCAHPC	Oakforest-PACS	DDN	IME	2048	2048	zip	275.65	492.06	154.41
4	Korea Institute of Science and Technology Information (KISTI)	NURION	DDN	IME	2048	4096	zip	156.91	554.23	44.43
5	DDN	IME140	DDN	IME	17	272	zip	112.67	90.34	140.52
6	DDN Colorado	DDN IME140	DDN	IME	10	160	zip	109.42	75.79	157.96
7	DDN	AI400	DDN	Lustre	10	160	zip	104.34	19.65	553.98
8	CSIRO	bracewell	Dell/ThinkParQ	BeeGFS	26	260	zip	88.26	67.44	115.50
9	KAUST	ShaheenII	Cray	DataWarp	1024	8192	zip	77.37	496.81	12.05
10	University of Cambridge	Data Accelerator	Dell EMC	BeeGFS	184	5888	zip	74.58	58.81	94.57

https://www.vi4io.org/io500/list/19-06/start

ISC’19 10-Node Challenge Results

#	information							io500		
	institution	system	storage vendor	filesystem type	client nodes	client total procs	data	score	bw	md
									GiB/s	kIOP/s
1	DDN Colorado	DDN IME140	DDN	IME	10	160	zip	109.42	75.79	157.96
2	DDN	AI400	DDN	Lustre	10	160	zip	104.34	19.65	553.98
3	University of Cambridge	Data Accelerator	Dell EMC	Lustre	10	320	zip	98.31	26.94	358.85
4	DDN	IME140	DDN	IME	10	160	zip	95.10	76.89	117.62
5	DDN Japan	AI400	DDN	Lustre	10	160	zip	74.10	12.22	449.28
6	HHMI Janelia Research Campus	Weka	WekaIO		10	3200	zip	66.43	27.74	159.12
7	CSIRO	bracewell	Dell/ThinkParQ	BeeGFS	10	160	zip	63.46	34.85	115.56
8	DDN	400NV	DDN	GRIDScaler	10	30	zip	59.49	13.55	261.21
9	WekaIO		WekaIO	WekaIO Matrix	10	700	zip	58.25	27.05	125.43
10	Oak Ridge National Laboratory	Summit	IBM	Spectrum Scale	10	160	zip	44.30	9.84	199.48

https://www.vi4io.org/io500/list/19-06/10node

Same Benchmark suite, but everyone limited to 10 Client nodes - fairer comparison between configurations

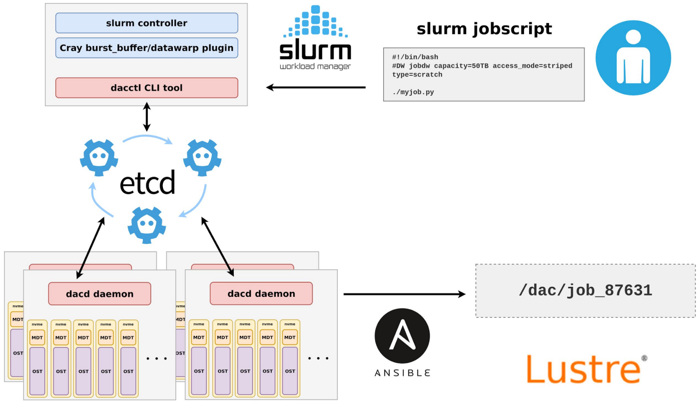
We actually had 10 nodes with dual-OPA multi-rail for this, boosting our bandwidth score - **Not normal on our cluster**

Overall Experience

- Lot of effort put in for just a benchmark, but useful learning experience for the team
  - Not often that we are so focused around performance outside of new-system provisioning
  - Was quite disruptive to our core cluster towards the end: re-imaging compute nodes to experimental Lustre builds required moving large portion of cluster out of active-service for intense couple of days of benchmarking
  - However the work helped us implement a number of network, troubleshooting and monitoring improvements that we are able to benefit from in future
  - Looking to apply some of what we learned for improving our more traditional disk-based scratch-tier
  - Provided interesting view into the latest features of Lustre. Things that we can learn from to guide design/improve future persistent-tier. Not often get to experiment with these things.
  - Fortunate to have a platform that provides ability to blow away and experiment with - ideal scenario for this kind of competition

Future Plans

- Focused now on improving the DAC software for our user-base
  - Working with one of our bigger users, SKA Science Data Processor team at Cambridge
  - They have already been using the DAC to find bottlenecks in their applications/workflows
  - Read/Writing lots of HDF5 files - benefitting from large bandwidth improvements over our current scratch
- Working on upgrading cluster to new 2.12.X Lustre baseline so can make use of latest features with the DAC in production
- Will aim to keep submitting to IO-500 in future, and also aim to start testing our traditional disk-based Lustre filesystems as a way to monitor performance of the filesystem over time



Future Plans - Probing DNE at larger scales?

- Have done some experiments looking at how 'mdtest' results scale with more MDTs
  - Used 24 (1 per server) for IO-500 submission, but what happens if we add more per server?

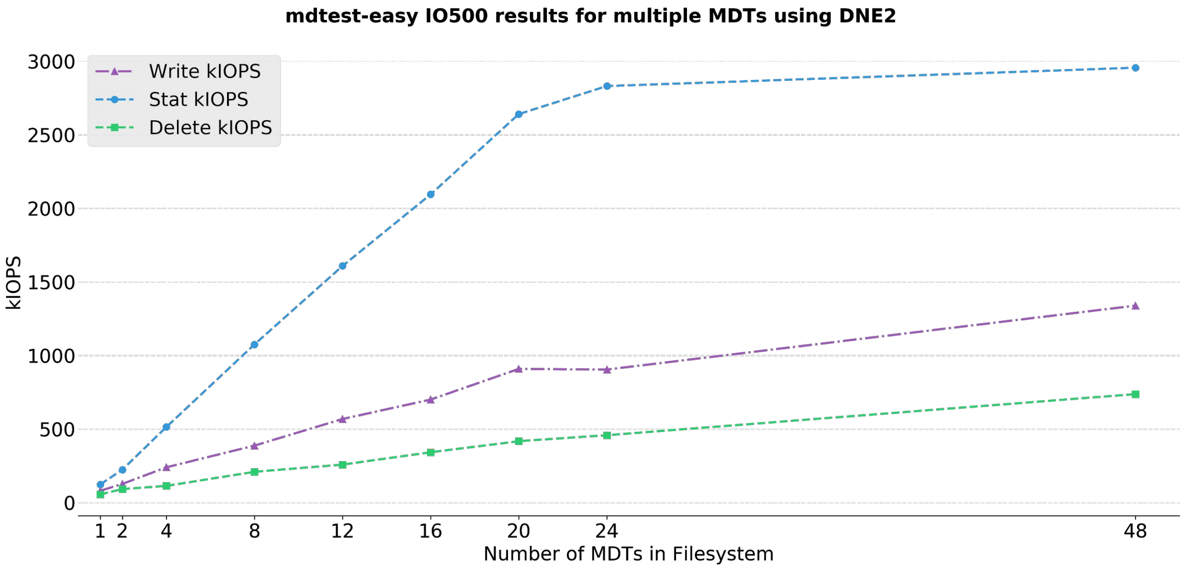
Have done some quick benchmarks with 2-per-server (48 MDTs) and saw further scaling, particularly for write/delete

Results shown here used our 'production' DAC configuration:

Servers: **Lustre 2.12.2**
Clients: **Lustre 2.10.7**

What about 3-per-server? (72 MDTs)

What about MDT on every device? (288 MDTs + DoM)



Acknowledgements

- Special thanks to all the engineers at Whamcloud for the help and support given. Particularly: Patrick Farrell, Shuichi Ihara, Amir Shehata, Peter Jones and Andreas Dilger

- Thanks to all my colleagues at Cambridge for support with the hardware and fabric

- And apologies to our users for the long queue times during the weekend around 8th June... 😀

Other Slides

Lustre version used

Base commit: `85db9b258c - New tag 2.12.54 (tag: v2_12_54)`

Patches applied:

```
647e37f LU-12043 llite: improve single-thread read performance           - https://review.whamcloud.com/#/c/34095
7dc9cfe LU-12043 llite,readahead: don't use max RPC size always          - https://review.whamcloud.com/#/c/35033
6967cf8 LU-11518 ldlm: control lru_size for extent lock                  - https://review.whamcloud.com/#/c/33371
d6f2913 LU-12325 dom: use LCK_PR with 'trylock' mode                     - https://review.whamcloud.com/#/c/35031
1f4f496 LU-11623 mdt: Opportunistically return UPDATE and PERM bits on open - https://review.whamcloud.com/#/c/33585
```

Commit reverted:

```
ce37c38691 LU-10213 lnd: calculate qp max_send_wrs properly
```

Workaround for an OPA issue encountered during testing - Tracked under [LU-12385] - Now resolved for 2.13

This was from a particular point in time - with expert support - not a recommendation of something to still use!

Recommended to start with latest feature-release, or ‘master’ if adventurous in testing