Additional I/O Considerations

Julian M. Kunkel

kunkel@dkrz.de

German Climate Computing Center (DKRZ), Research Group/Scientific Computing (WR)

04-07-2016

Outline

I/O Architecture (Phase 1)

- 31 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
- 31 Extension units (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

I/O Architecture (Phase 2)

- Additional file system (Now two file systems in total)
  - Mounted on all compute nodes
  - Characteristics: 11 k disks, 52 PB storage
- 34 ClusterStor L300 Scalable Storage Units (SSUs)
- 34 Extension units (JBODs)
- Storage hardware
  - Seagate Enterprise Capacity V5 (8 TB) disks
- Multiple metadata servers
  - Root MDS + 7 DNE MDS

Parallel File System

Lustre 2.5 (Seagate edition, some backports from 2.7+)

Filesystem

- We have two file systems: /mnt/lustre0[1,2]
- Symlinks: /work, /scratch, /home, ...
- For `mv`, 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)
- We will transfer some projects to the phase 2 file system

Peak Performance

Phase 1 + 2

- 65 SSUs · (2 OSS/SSU + 2 JBODs/SSU)
- 1 Infiniband FDR-14: 6 GiB/s ⇒ 780 GiB/s
- 1 ClusterStor9000 (CPU + 6 GBit SAS): 5.4 GiB/s
- L300 yield IB speed, still we consider 5.4 GiB/s ⇒ aggregated performance **704 GiB/s**
- Phase 2: `obd-filter` survey demonstrates that 480 GB/s and 580 GB/s can be delivered

Performance Results from Acceptance Tests

- Throughput in GB/s (% to peak) measured with IOR
  - Buffer size 2000000 (unaligned) on 42 OSS (Phase 1) and 64 (P 2)
  - In the phase 2 testing, the RAID of at least one OSS is rebuilding

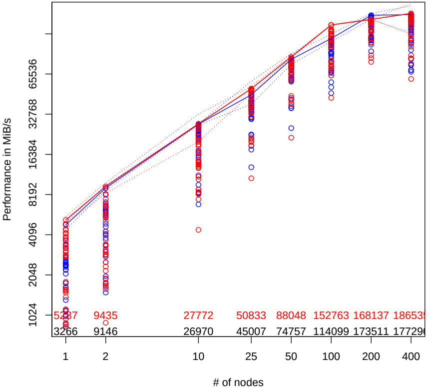
	Phase 1		Phase 2	
Type	Read	Write	Read	Write
POSIX, independent[1]	160 (70%)	157 (69%)	215 (62%)	290 (84%)
MPI-IO, shared	52 (23%)	41 (18%)	65 (19%)	122 (35%)
PNetCDF, shared	81 (36%)	38 (17%)	63 (18%)	66 (19%)
HDF5,shared	23 (10%)	24 (11%)	62 (18%)	68 (20%)
POSIX, single stream	1.1 (5%)	1.05 (5%)	0.98 (5%)	1.08 (5%)

- Metadata measured with Parabench
  - Phase 1: 80 kOPs/s
    - 25 kOP/s for root MDS; 15 kOP/s for DNEs
  - Phase 2: 210 kOPs/s
    - 25 kOP/s for root MDS; 30-35 kOP/s for DNEs

[1] 1 stripe per file

Performance with Variable Lustre Settings

- Goal: Identify good settings for I/O
- IOR, indep. files, 10 MiB blocks on Phase 1 system
  - 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

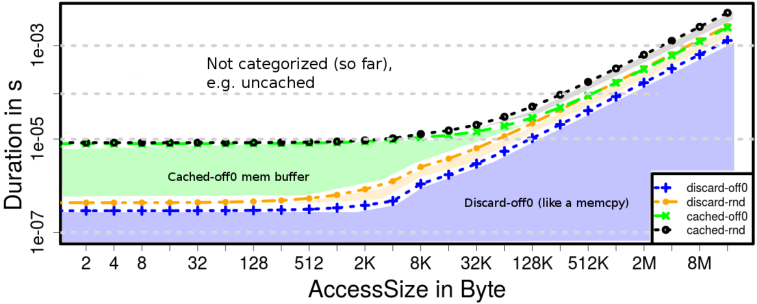


Best settings for read (excerpt)

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

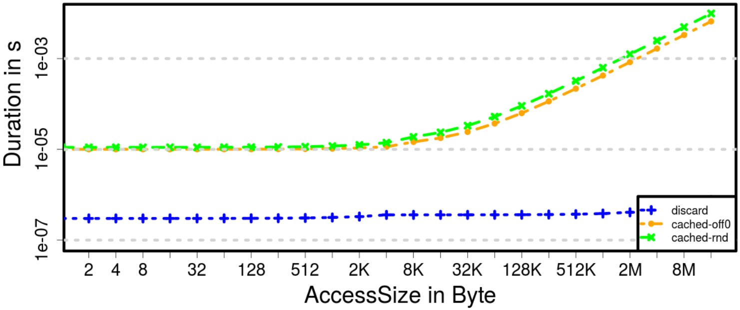
I/O Duration with Variable Block Granularity

- 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

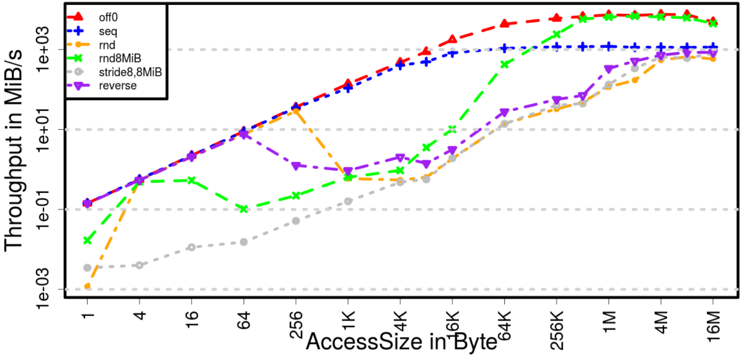


Write

- Memory layout has a minor impact on performance

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

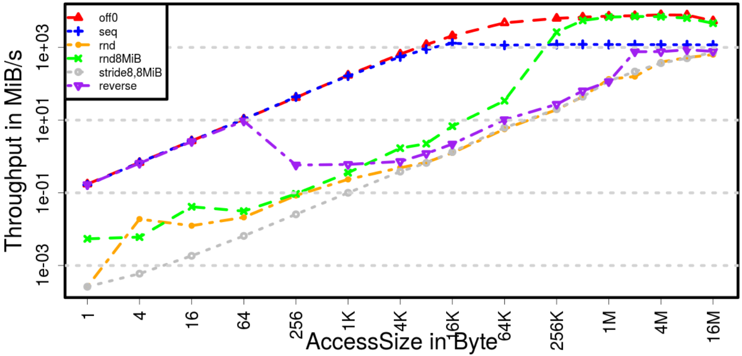
Throughput with Variable Granularity



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



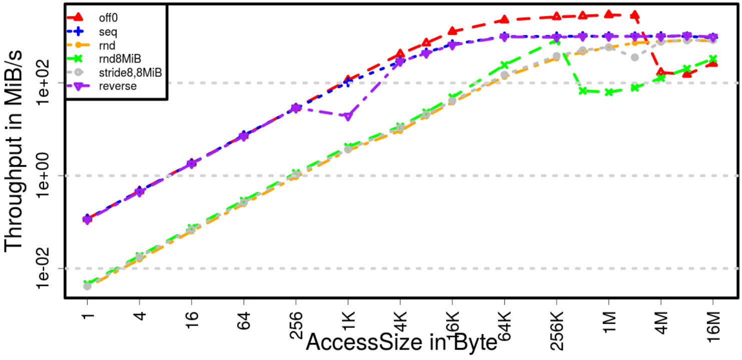
Read – clean cache

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

Potential Solution to Caching: FUSE

- Utilizes page cache
- Measurements: 4k reads: 100k ops on Lustre, 3 M with FUSE
- FUSE3 client implemented that allows to re-mount subtree
- Not cache-coherent, but does not matter for certain workloads
  - /sw (loading Python modules takes seconds)
  - configure/compilation
- Implementation complete, we have to check it on test system...

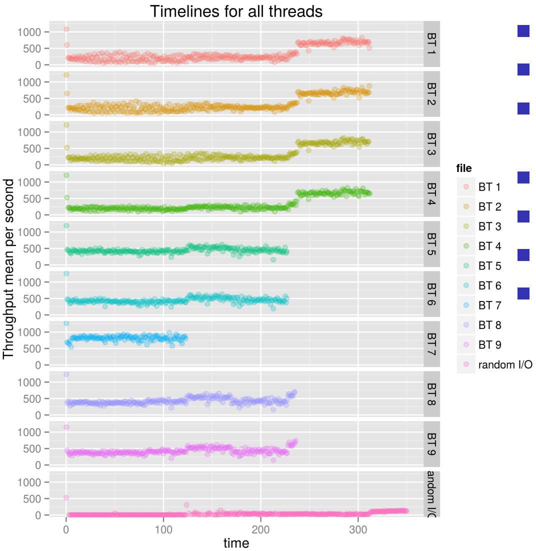
Throughput with Variable Granularity



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



- Storage == shared resource
- Independent file I/O on one OST
- Running 9 seq. writers concurrently (10 MiB blocks)
- One random writer (1 MiB blocks)
- Each client accesses 1 stripe
- Each client runs on its own node
- Observations
  - BT: 3 performance classes
  - RND without background threads: 220 MiB/s
  - RND with 9 threads: 6 MiB/s
  - Slow I/O dominated by well-formed I/O
  - Reason: IB routing

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.

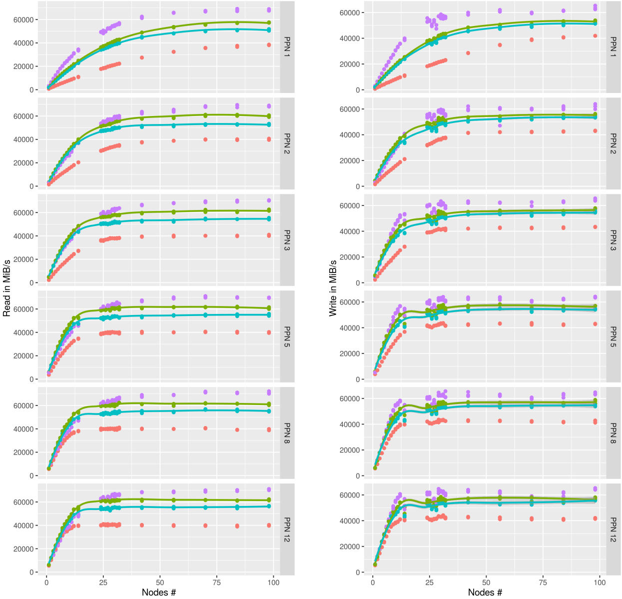
Outline

In-Memory Storage

- Kove XPD
  - Offers various interfaces to access data
    - KDSA, malloc, mmap, block device
  - Use case: burst buffer, in-situ, memory-extension?
  - Memory-extension: HLRE4 at DKRZ, only buy one type of system (2 GiB Mem/Core)?
    - Rest could be dynamically provisioned, i.e. at runtime assign (shared) memory !
    - I would love to see a discussion between DKRZ, Bull and Kove [2]
- Benchmarking of Kove XPD
  - In memory I/O
  - Persists data onto 24 HDDs
  - Takes 10 min to synchronize system (under full load)
- Three devices with 6+4+4 = 14 IB links
  - Peak performance: 70 GiB/s
- Created an MPI-IO wrapper to their KDSA library
- Benchmarked random I/O with IOR
  - Sequential behaves similarly (!)

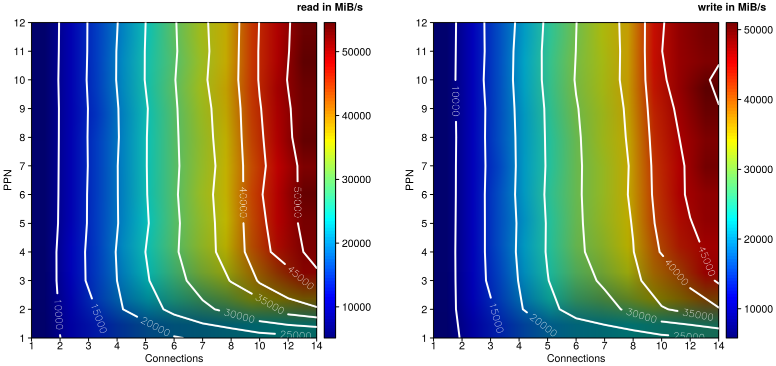
[2] john.overton@kove.net

Varying Client Node Count, PPN, Block Size

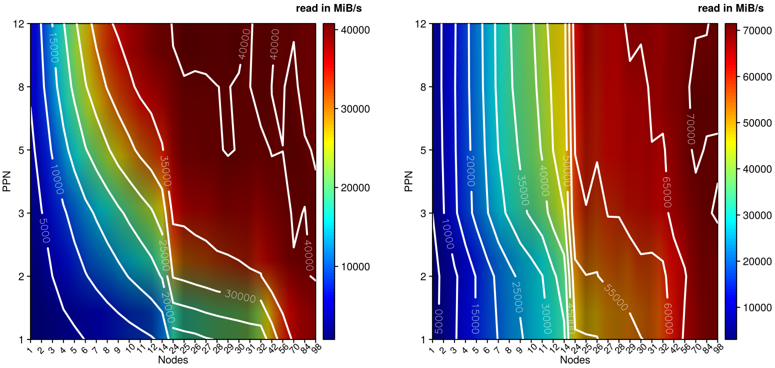


Varying Number of Connections

- 100 KB accesses
- 14 nodes

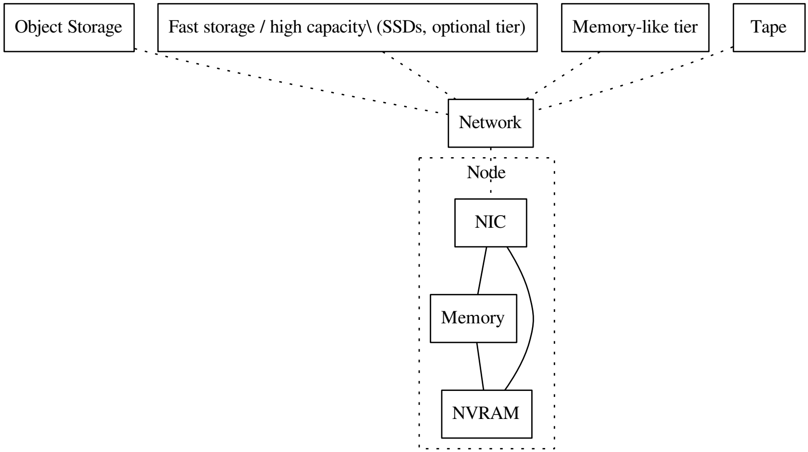


Performance Map for Reads



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

Future Storage



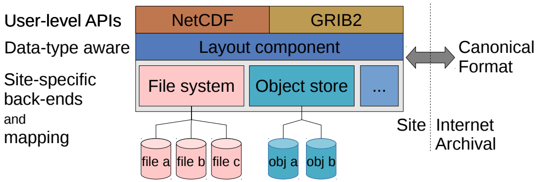
Storage in ESiWACE

H2020 project: ESiWACE Center of Excellence

Work package 4

Partners: DKRZ, STFC, ECMWF, CMCC, Seagate, (HDFGroup)

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



Current Approach within ESiWACE/WP4

- We started to implement a HDF5 in-memory VOL
- First tests with a NetCDF benchmark are functional
- We will extend the current prototype to test various backends
  - Store scientific metadata in (non-)SQL DBs
  - Utilize backends for data only
    - DDN (WOS/IME to follow), Seagate, file system
- We will work on models for our next systems

Other projects

Intel Parallel Computing Center (for Lustre)

- Implementation of client-server compression
- Will also improve Lustre throughput for uncompressed I/O
- Testing of client-side extensions on Mistral planned!

AIMES

- DSL & I/O for ICO models
- User-defined/workflow oriented lossy compression
- Little bit of optimization for HDF5/NetCDF

Summary

- Lustre performance behavior is suboptimal (not limited to Seagate's edition)
  - Pending features will improve the situation (e.g. dynamic striping)
- Next system at DKRZ
  - Let us try to go away from parallel file systems (and POSIX)
  - I hope we get away from Lustre (with its tight kernel integration)
- There are quite some activities ongoing within our research group
- Options to evolve the relationship in terms of I/O activities
  - Modelling of (performance and costs) of future systems
  - Improving scientific file formats

My 5 Cents

- Scientific productivity is the goal
- Future systems will change the way we use them for HPC
- We will be able to run legacy applications
  - Maybe at 5% what is possible with novel workflows
- Managing and accessing I/O will definitely change
  - Too many prototypes are already in production and more to come
- Standards across data centers are needed
  - Consortia to define and implement (storage, montioring etc.) APIs
- Need for separation of concern between domain scientists, scientific programmer, system architect and computer science
  - Increase the abstraction level, decouple code

Laboratory for I/O Investigation

Virtual Lab: Conduct what if analysis

- Design new optimizations
- Apply optimization to application w/o changing them
- Compute best-cases and estimate if changes pay off

So far: Flexible Event Imitation Engine for Parallel Workloads (feign)

- Helper functions: to pre-create environment, to analyze, ...
- A handful of mutators to alter behavior
- Adaption of SIOX is ongoing to allow on-line experimentation

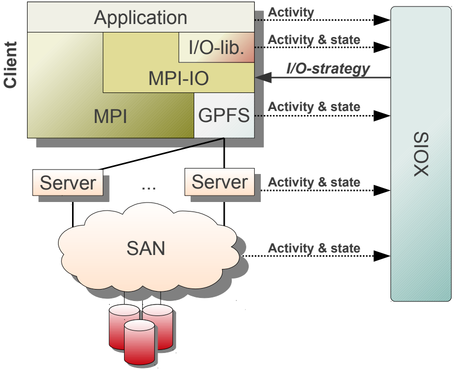
Additional Research @ WR

- Compression of scientific data
  - Lossless (1.5:1 to 2.5:1)
  - Lossy: rate 12:1 to 50:1[3]
  - Interfaces for specifying tolerable loss
- Domain-specific languages
  - Retain code-structure
  - Improve readability
  - Intelligent re-structuring of code at compile time
- Alternative interfaces, usage of object storage
- Monitoring
- We push (computer science) standards towards the needs of scientists

[3] WaveletCompressionTechniqueforHigh-ResolutionGlobalModelDataonanIcosahedralGrid,Wanget.al,2015

Scalable I/O for Extreme Performance (SIOX)

Started as collaborative project between UHH, ZIH and HLRS



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

SIOX Ongoing Work

Automatic assessing the quality of the I/O

```
Your Read I/O consisted of:
200 calls/100 MiB
10 calls/10 MiB were cached in the system’s page cache
10 calls/20 MiB were cached on the server’s cache
100 calls/40 MiB were dominated by average disk seek time (0.4
...
5 calls/100 KiB were unexpected slow (1.5s time loss)
```

Follow up Project

- Together with our partners we submitted a follow up project
- To increase scalability and assessment capability