The NEXTGenIO project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 671951

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

echofs: A scheduler-guided temporary filesystem to leverage node-local NVMs

Alberto Miranda, Ramon Nou, Toni Cortes

{alberto.miranda, ramon.nou, toni.cortes}@bsc.es

Lyon Sept. 2018

nextgenio

SBAC-PAD 2018

I/O in HPC -> a fundamental challenge

Petascale struggles with HPC I/O:

Extreme parallelism [+10^6 threads]

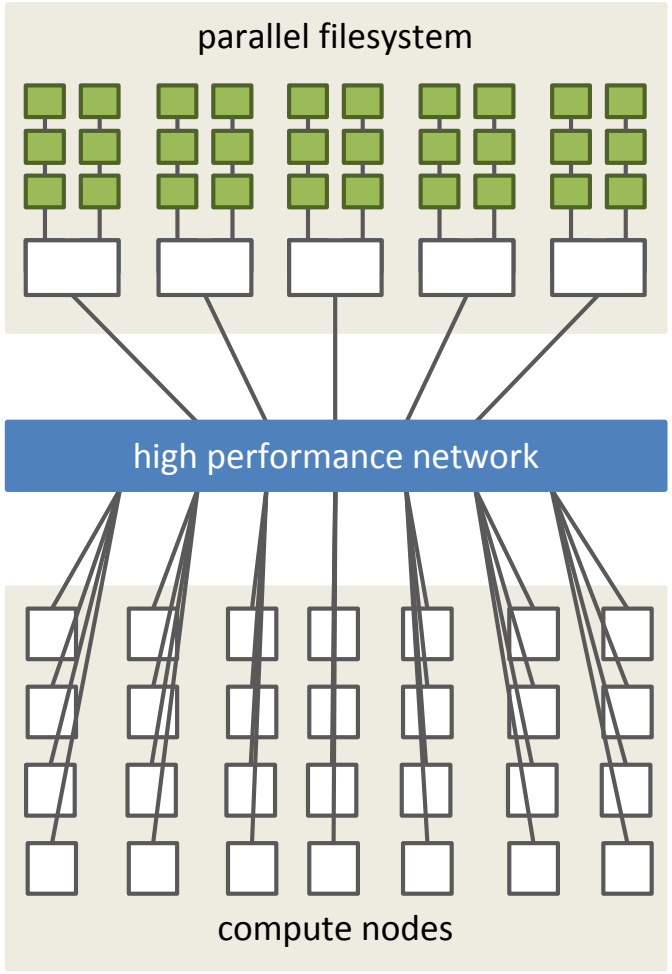
Large input files [+1TiB]

Periodical checkpointing

Large outputs [big files/lots of small files]

Data Analytics coming to HPC

Exascale will be worse

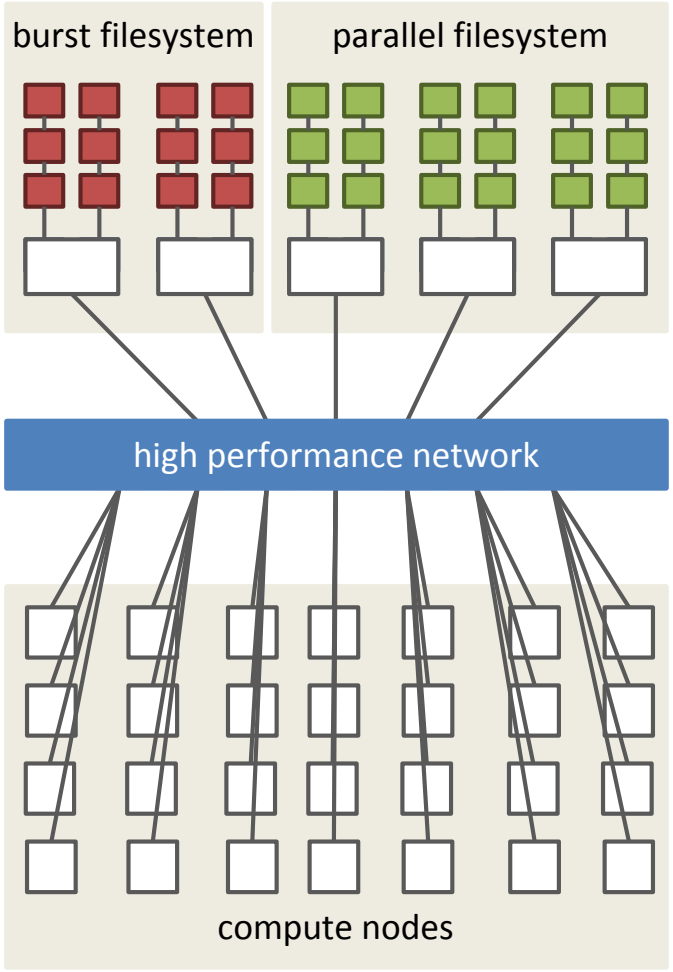


Burst buffers

Burst buffers: Fast storage devices to temporarily store data before PFS

Remote: separate, dedicated HW & interconnect fabric

Cray® Datawarp™, DDN IME®



Burst buffers

Burst buffers: Fast storage devices to temporarily store data before PFS

Remote: separate, dedicated HW & interconnect fabric

Cray® Datawarp™, DDN IME®

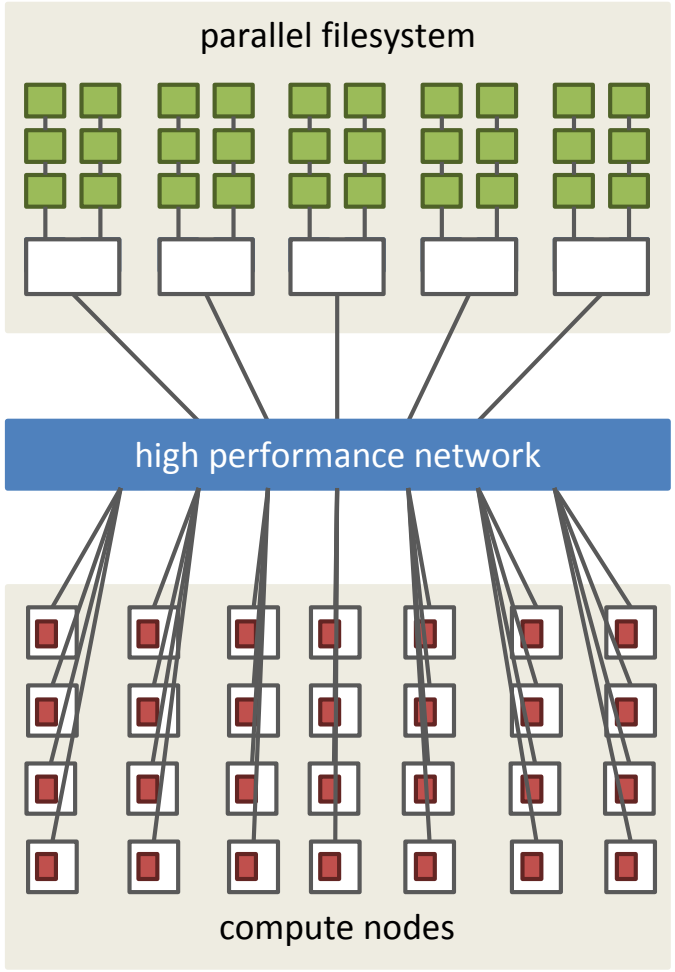
Node-local: using compute node resources

Argonne's Theta: 128GB SSDs/node

BSC's Marenostrum IV: 240GB SSDs/node

Lots of technologies: HDD, SSD, NVM

Usage, allocation and management of data falls upon users



echofs: Goals

Allow applications to benefit transparently from new storage (e.g. NVM)

Storage layers unified under mount point

I/O stack complexity hidden

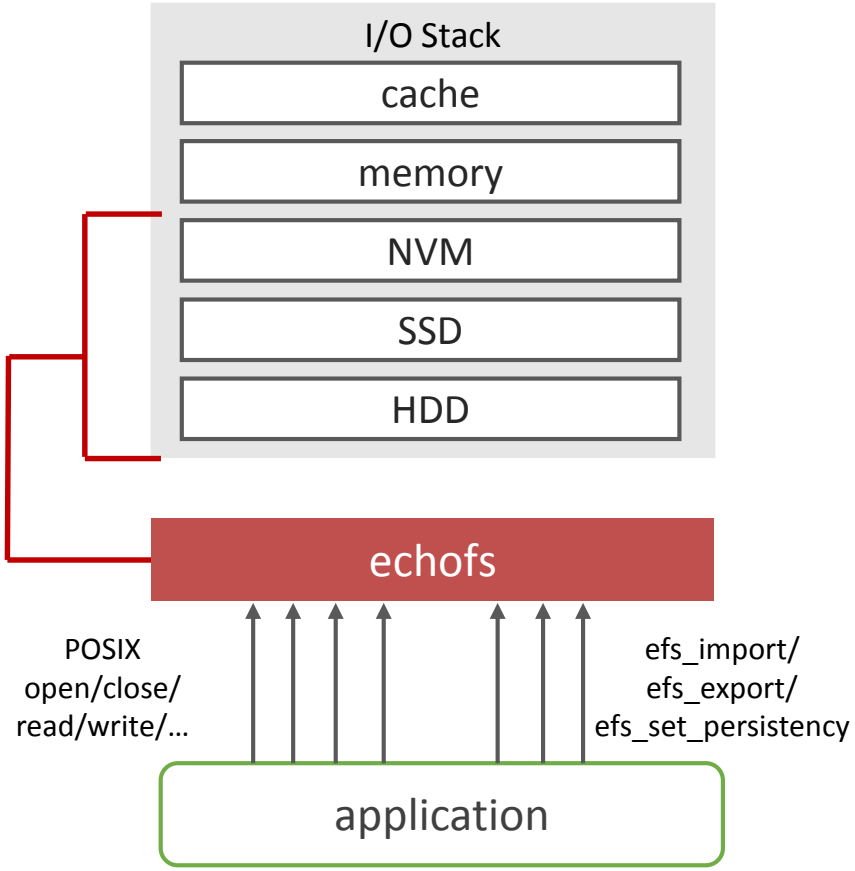
Use information to reduce PFS contention

Coordinate w/ batch job scheduler to transform **random job I/O** into **well-ordered import/export I/O**.

Capture application's data dependencies

Allow apps to "tag" **input/output** and **temporary/persistent/reusable** files

Allow apps to request data transfers from/to PFS



echofs: Workflow

User provides job I/O dependencies through job scheduler (e.g. SLURM)

Nodes required, files accessed, access type

Scheduler gets nodes and mounts echofs

Filesystem **imports input data** from PFS and creates a **POSIX namespace**

Scheduler starts job

Data accessed through echofs mount point

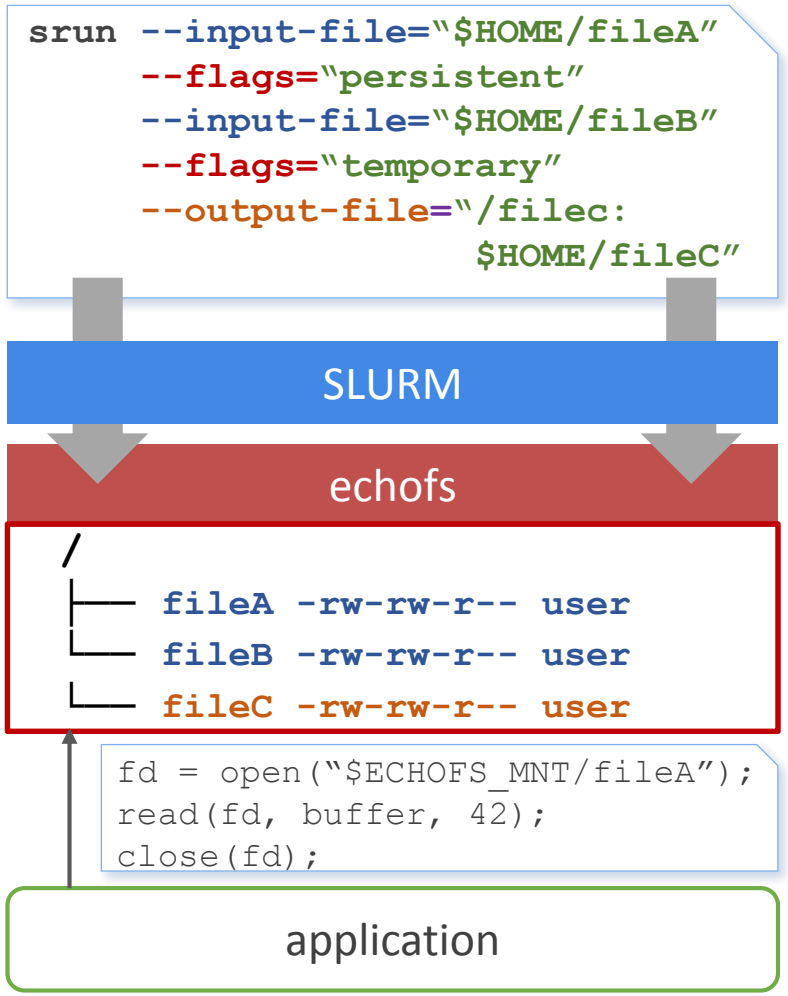
New data created into **local storage**

echofs destroyed when job ends

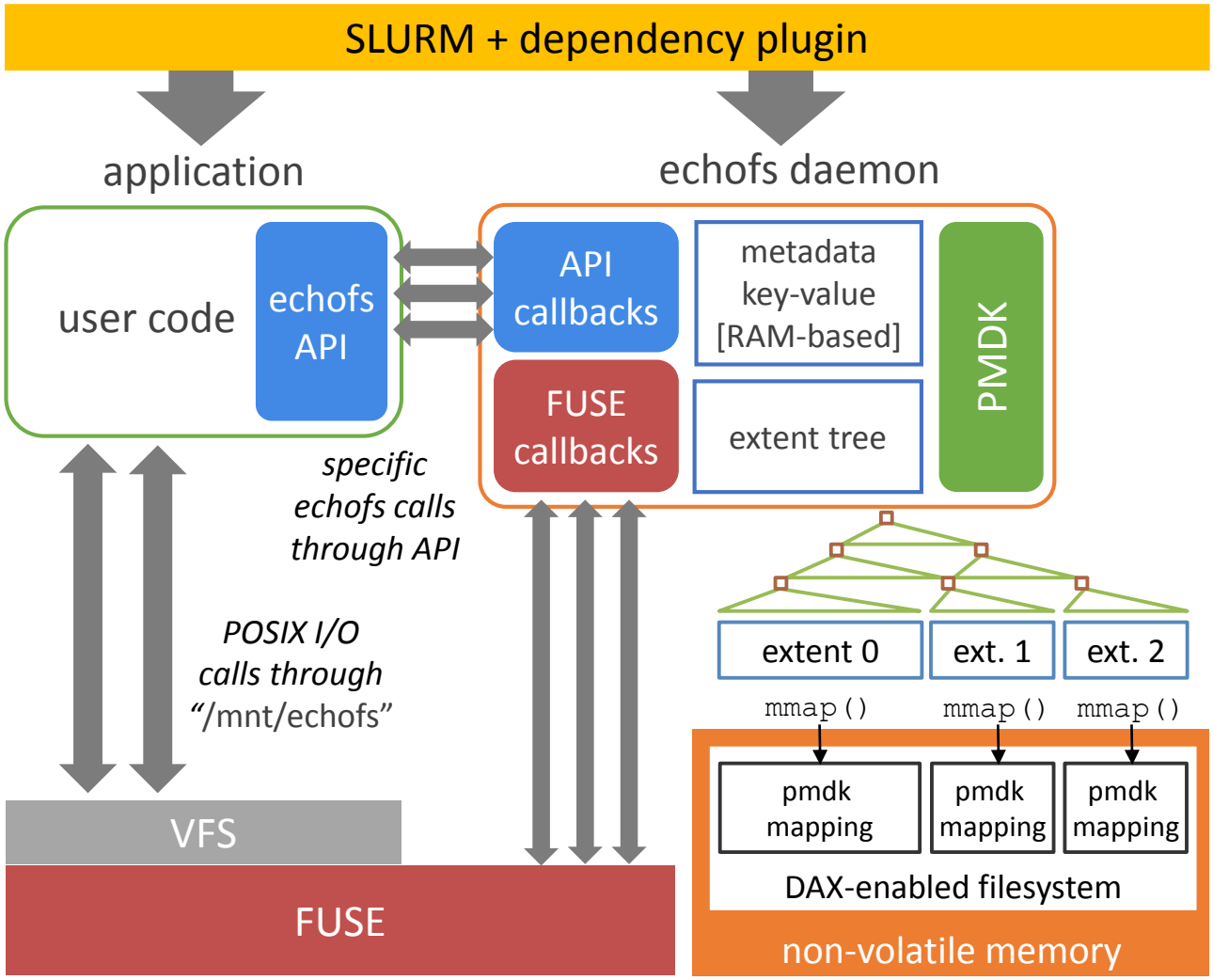
Persistent files sent to PFS

Temporary files deleted

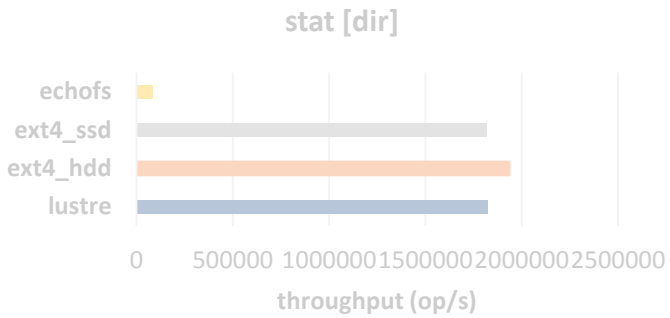
Reusable files left in node

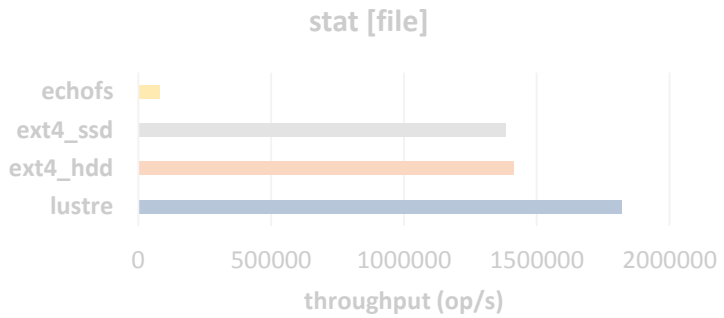


echofs: Architecture



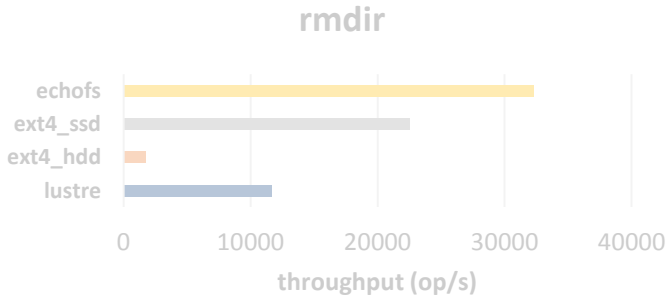
Experiments [metadata performance]

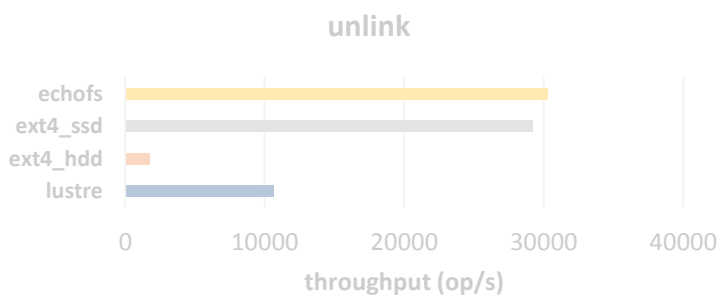




MDTEST microbenchmark
2*10E6 files/dirs x single parent
4 concurrent processes
10 iterations

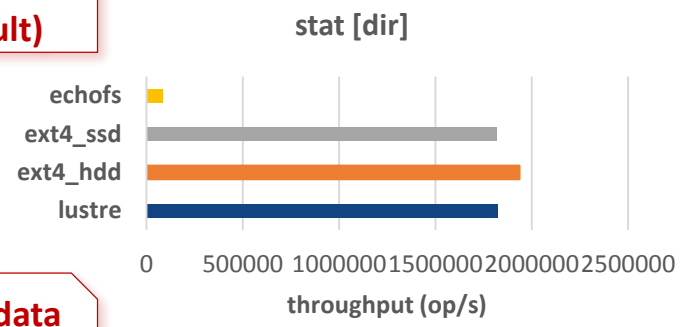
development cluster: 10 nodes
Lustre FS: 1 MDS, 2OSTs, SSD backend

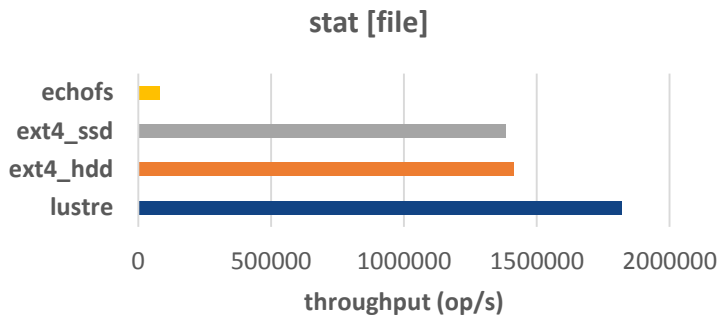




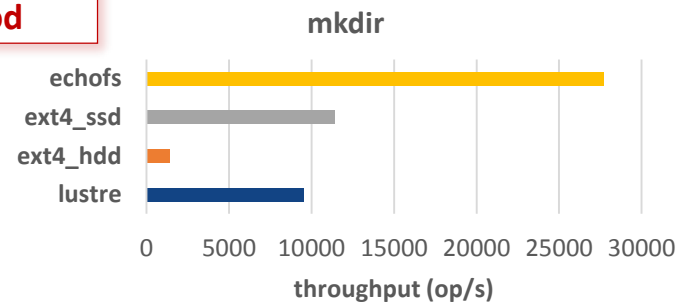
Experiments [metadata performance]

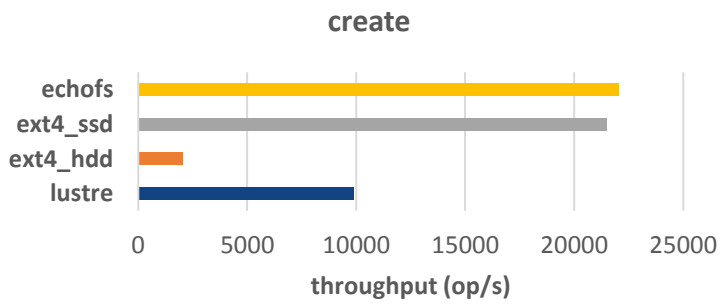
stats are slow (FUSE's fault)

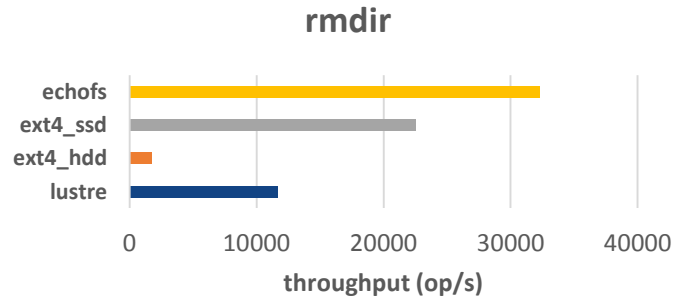


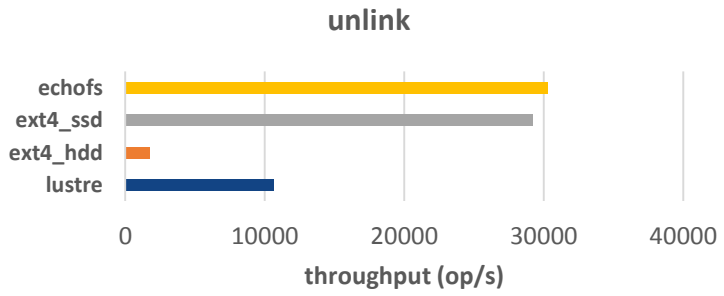


other metadata operations are quite good

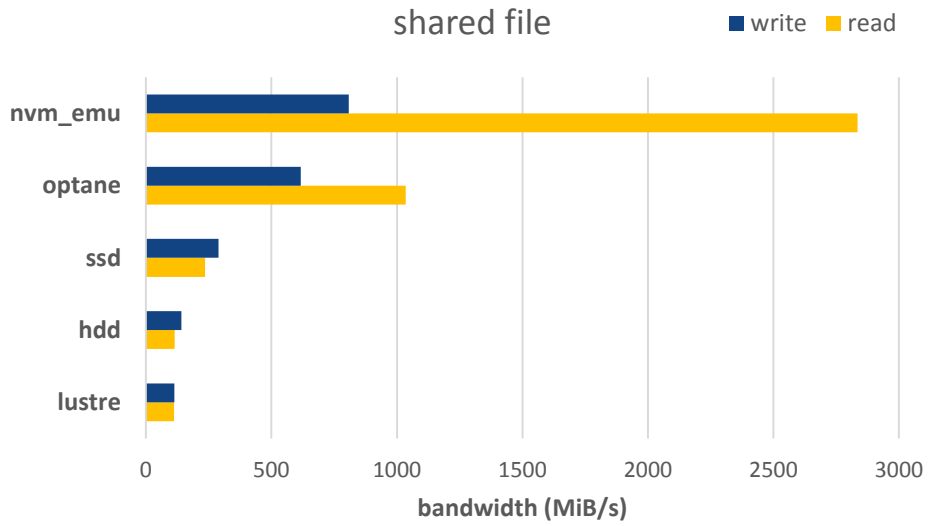




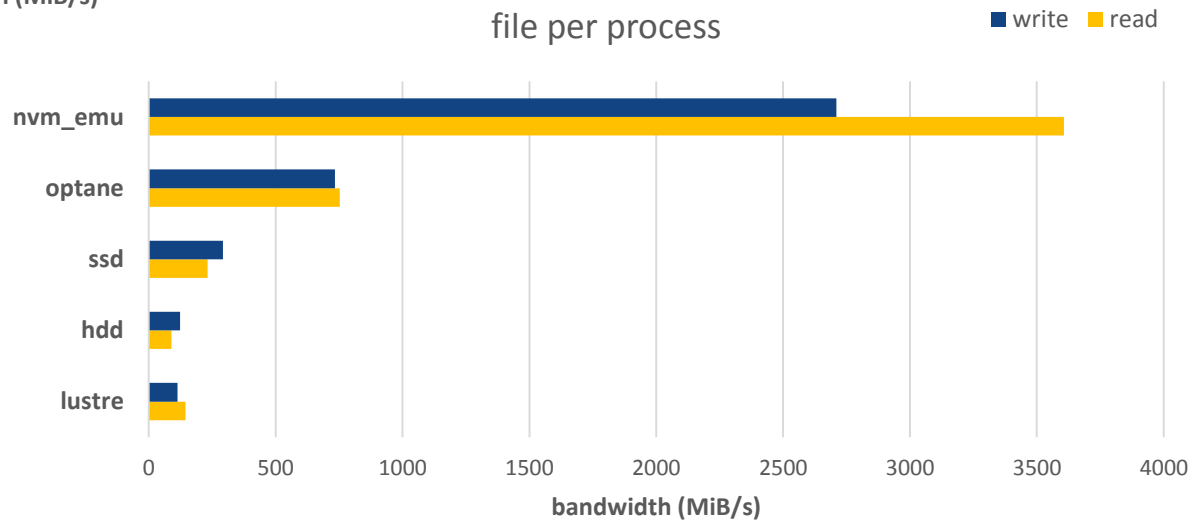




Experiments [data performance]

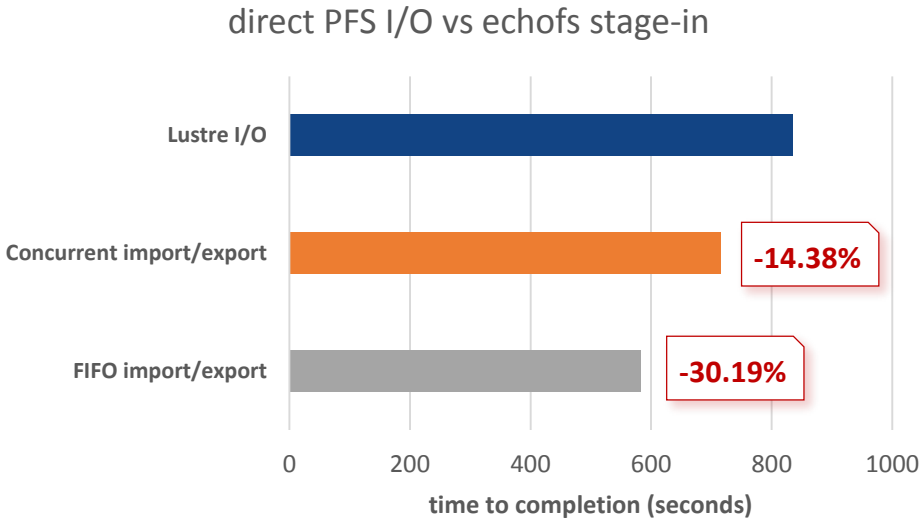


IOR microbenchmark
1GiB data (single/multiple files)
128KiB transfer size
4 concurrent processes
10 iterations



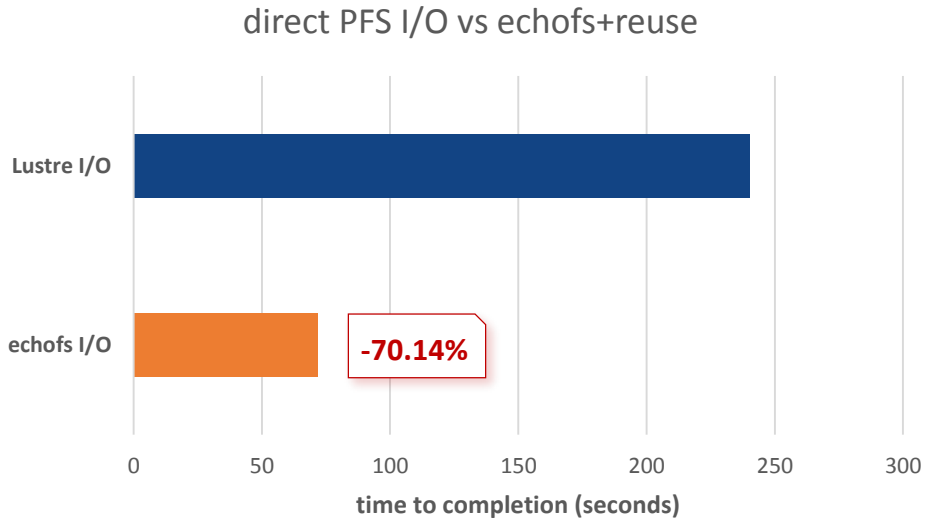
despite FUSE's limitations, data performance is significantly better than accessing Lustre

Experiments [scheduler coordination]



3x16 GiB files created in Lustre
full read + full rewrite workload
3 nodes, 10 iterations

2x2.5 GiB files created in Lustre
full read + full rewrite workload
2 nodes, 10 iterations
echofs + reuse



Conclusions

The performance-advantage of node-local burst buffers can help to "containerize" application I/O

Less I/O to the PFS means less overall cluster contention → faster applications → happier users

Changing application I/O into import/export workflows helps the PFS (a lot)

I/O is no longer perceived as random accesses

Prefetching mechanisms in the PFS can be more effective

Opens the door for effective **data-aware scheduling**

Application-specific filesystems don't need to be full-fledged filesystems to achieve gains

Example: Not all applications need full POSIX compliance