# IO Benchmarks

Jesus Espinoza, Luca Bonaldo, Eduardo Quintana,
Federico Barone and Herbert Nguruwe.

**Exercise 1 - NetCDF**

The performance of the **Netcdf** library for Fortran has been benchmarked in the Lustre file system of the C3HPC cluster. This benchmark consists of the execution of 4 Fortran codes each one spanning 8 MPI processes. Two of the aforementioned codes, `collect_and_write.F90` and `read_distribute.F90`, perform write and read operations using the spokesperson approach. While the other two, `allwrite.F90` and `allread.F90`, perform those operations in parallel using the syntactic ease of **Netcdf**. The results of the benchmark are show in pictures 1 and 2.

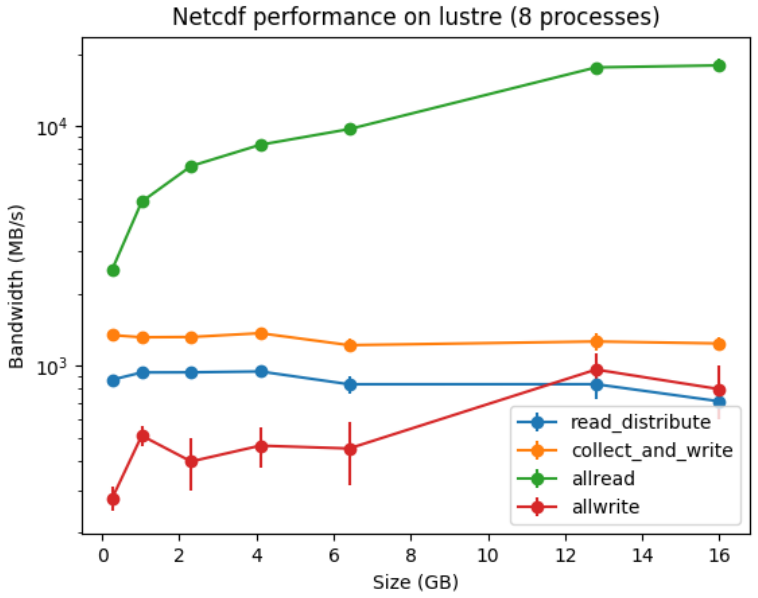



Figure 1: Bandwidth obtained for the codes at different global data sizes

In both cases the `allread` code, reaching bandwidths of the order of 10 GB/s, outperforms the physical capabilities of the OST servers, which for a stripping factor of 1 cannot exceed 400 MB/s. The explanation to this behaviour is that the target data, once written into disk, it remains cached in RAM and after the consecutive read request it is read from the latter faster device.

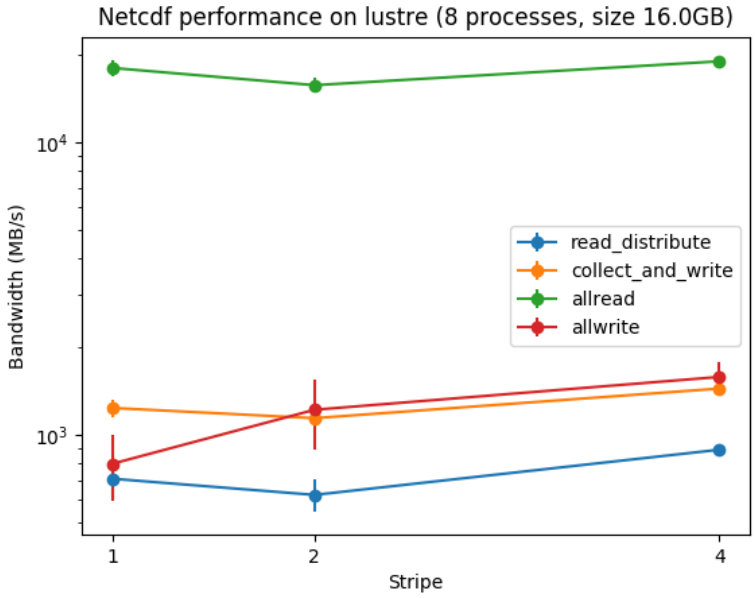



Figure 2: obtained for the codes for a fixed global data size and while stripping the target file in 1, 2 and 4 pieces.

On the other hand, writing bandwidths seem to be legitimate. These show that without the stripping factor, the `allwrite` program is outperformed by `collect_and_write`, possibly due to overhead of the different software layers from which Netcdf is made off. This fact persists in spite of the spokesperson approach's bandwidth remaining almost constant independent of the data size, while the parallel writing (`allwrite`) gains significant performance for 10 GB data sizes. For striped files, the situation is reversed: having parallel writing performing slightly better than `collect_and_write`.

**Exercise 2 - IOzone**

In this exercise we ran the IOzone benchmark for both Ulysses and C3E. We measured the read and write bandwidth for different file sizes (8GB, 64GB, 128GB, 150GB) on 4 OST (2 OSTs for Ulysses), one at a time. We omitted the results for reading an 8GB file because we measured a bandwidth of 4GB/s probably due to the fact that it was buffering the file in the RAM. Figure 3 shows the bandwidth measurements for writing. Figure 4 shows the results for reading.

From the results it can be noted that, as expected, the bandwidth is approximately of the same order for all the OSTs in both reading and writing.

C3HPC has a read bandwidth that is twice the one of Ulysses. For writing, C3HPC has a bandwidth that is approximately 5 times the one we measured in Ulysses. This is probably because Ulysses was having some problems with the IO Servers during the time we were running this benchmarks.

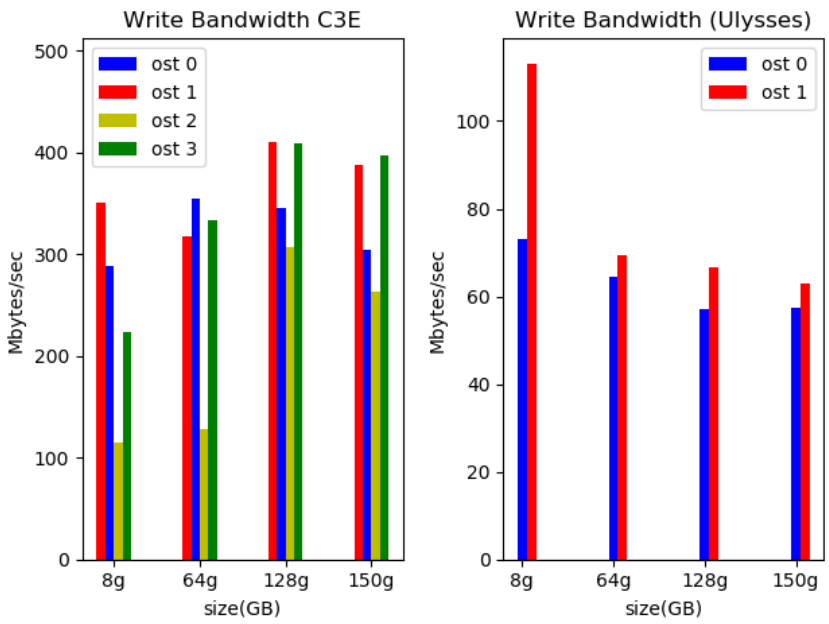



Figure 3: Writing bandwidth for different OSTs on C3E cluster (left) and Ulysses cluster (right). We used 3 different file sizes. The block size was of 1 MB.

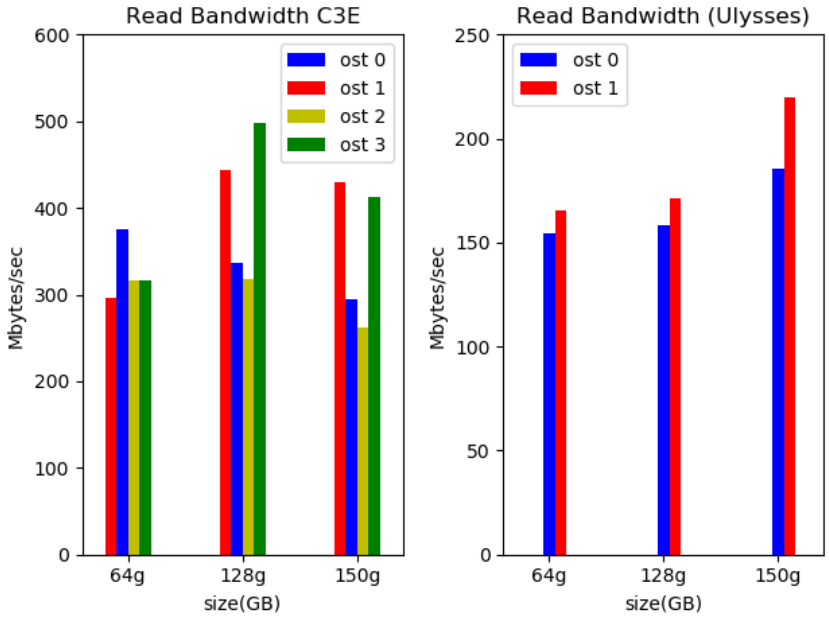
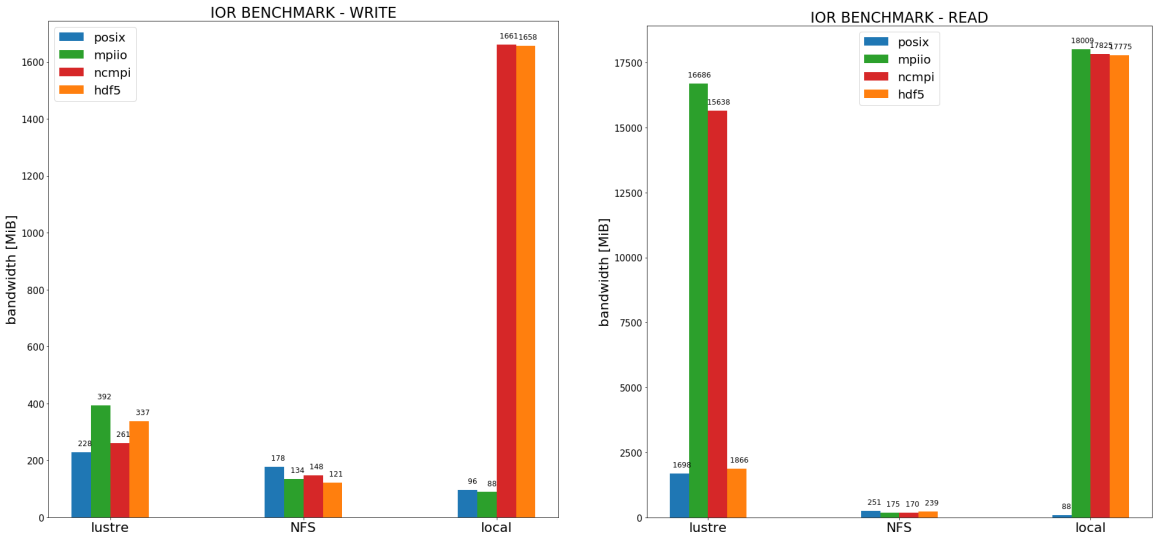



Figure 4: Reading bandwidth for different OSTs on C3E cluster (left) and Ulysses cluster (right). We used 3 different file sizes. The block size was of 1 MB.

**Exercise 3 - IOR**

For this exercise we were asked to run the IOR benchmark to measure the bandwidth for all




(a) *Bandwidth in writing* (b) *Bandwidth in reading*

Figure 5: Bandwidth in writing (left) and reading (right) using 4 MPI processes in the C3E cluster. The aggregated file size was 4 GB.

the file systems present in the C3HPC cluster:

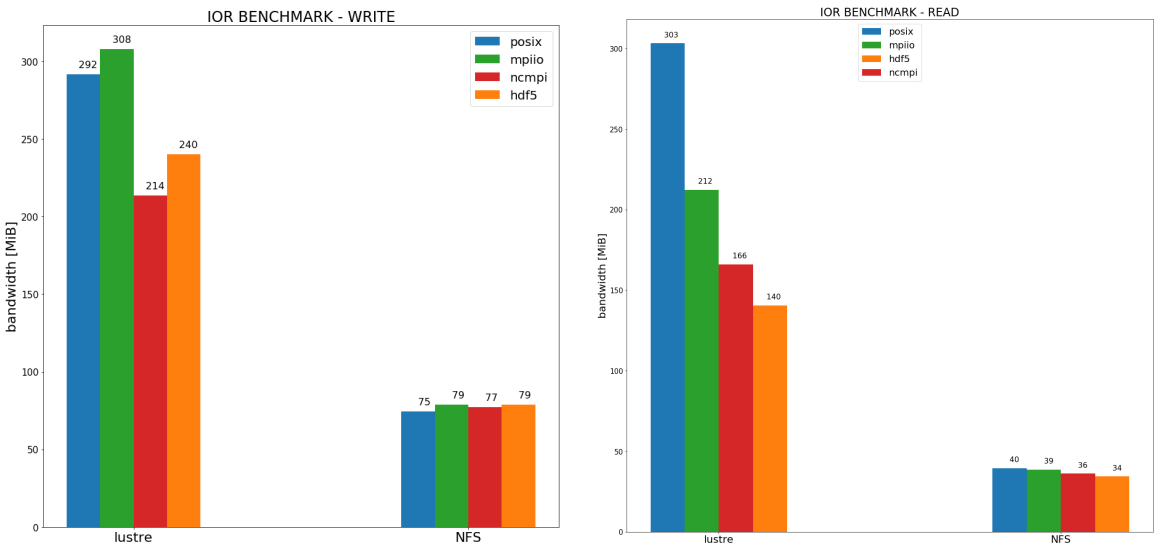
- Lustre: We write/read through the IO Servers to the OSTs.
- Network File System (NFS): We write/read in the disks in the login node.
- Local File System: We write/read in the local disks inside the node.

using the following APIs: `POSIX`, `MPI-IO`, `HDF5` and `NCMPI`.

### Benchmarks without striping

As an initial test of the benchmark, we performed several runs on the C3HPC cluster using 4 MPI processes which read and wrote a file in parallel with a blocksize of 1MB. The figure 5 shows the obtained results.

Afterwords we ran the benchmark twice, using 48 simultaneous MPI processes, each one writing/reading a segment of 3 GB of a single file. We choose this segment size in order to avoid buffering in the RAM, which can lead to unreliable results; being this an appropriate explanation for the difference between the first set of benchmarks and the second one. For both tests, the block size was of 1 MB. For this part of the exercise we didn't stride the file for the Lustre File System. We will treat this particular case in the following section.
From the plot 6 we can see that the Lustre file system outperforms the NFS file system. This is due to the fact that Lustre has a better network bandwidth plus it uses a RAID array to store the data. Keep in mind that we are not striping the file among different OSTs, which can potentially boost performance even more, as we will see next.
Furthermore, regarding the four APIs, `POSIX` and `MPI-IO` show the best results in both writing and reading for Lustre, while in the NFS file system the four APIs have a comparable performance.




(a) *Bandwidth in writing* (b) *Bandwidth in reading*

Figure 6: Bandwidth in writing (left) and reading (right) using 48 MPI processes in the C3E cluster. The aggregated file size was 146 GB.

### Benchmarks on Lustre with striping

The second part of this exercise consisted on measuring the impact of striding a file for writing/reading using the Lustre File System. Figure 7 show our results. For this benchmark we used 48 processors writing/reading simultaneously on a single file of size 48 GB. The block size was of 1 MB.

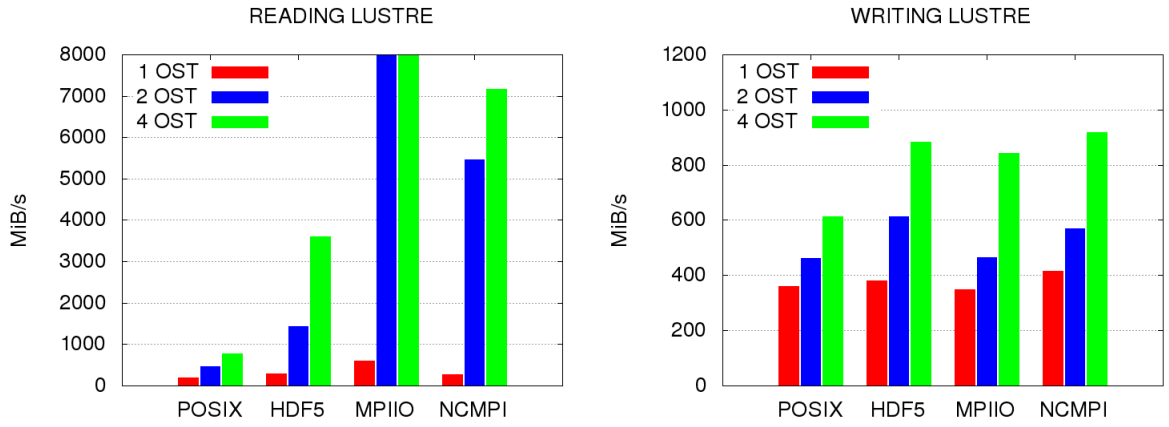



Figure 7: Reading bandwidth for different OSTs on C3E cluster (left) and Ulysses cluster (right). We used 3 different file sizes. The block size was of 1 MB.

As expected, as we increased the number of OSTs used we noticed a significant increase in the bandwidth for both, writing and reading. Values are out of scale for `MPIIO` because it

is probably using the the RAM as a buffer.

### Benchmarks using POSIX on Lustre: N → 1, N → N

Lastly we tested the difference between running the IOR benchmark using `POSIX` writing/reading to/from a single file vs. to/from N files, were N is the number of MPI processes. For this test we use 48 cores, an aggregated file size of 146 GB and a block of size 1 MB. For both cases we collected data from two runs. Table 1 show our results.

| Comm | Writing BW [MiB] | Reading BW [MiB] |
|---|---|---|
| N to 1 | 341.67 | 297.03 |
| N to N | 748.45 | 673.85 |

Table 1: Writing and reading bandwidth for POSIX on Lustre using an N to N pattern vs an N to 1 pattern.

These results were expected: N to N has a better performance than N to 1 since the processes write each file independently, thus avoiding the locking mechanism of POSIX.

### Exercise 4 - Darshan

As a last exercise, the darshan tool has been used to profile the execution of the IOR benchmark using POSIX HDF5 MPIIO as APIs. In order to do that it is necessary to link the darshan library during the run of benchmark which allows to obtain a PDF with the analysis of the I/O operations in terms of performance, memory usage, execution time, bandwidth and other important parameters. Since nowadays the I/O operation are becoming the bottleneck of many applications and system, an analysis in terms of the I/O operations result crucial if we want to obtain an high-performance code. In the darshan folder we collect the obtained PDFs for the three APIs used to run the benchmark using 48 cores each one writing/reading a segment of 3 GB with a block of size 1 MB.