"Expand: High Performance Storage System for HPC and Big Data Environments" (TED2021-131798B-I00)

High Performance Storage Sytems for HPC and Big Data (Expand)

D 3.2
Report on Final Evaluation

Universidad Carlos III de Madrid

June, 2025

CONTENTS

1. RESULTS FOR HPC BENCHMARKS

This chapter presents the results of Expand evaluations using different benchmarks in the HPC4AI Laboratory supercomputing cluster, described in Section 1.1. The results when Expand does not use fault tolerance and malleability will be shown in Section 1.2, the results when using fault tolerance will be presented in Section 1.3, and the results when using malleability will be introduced in Section 1.4.

1.1. HPC Environment

To evaluate Expand presented in this chapter, the HPC4AI Laboratory [1] supercomputing cluster of the University of Turin (UNITO) will be used. The specifications of this HPC environment can be found in Table 1.1.1.

Table 1.1.1: HPC environment specifications.

Attribute	**HPC4AI *Laboratory*** [1]
Number of nodes	68 nodes
Node vendor	Lenovo NeXtScale nx360 M5
CPU per node	2 x Intel Xeon E5-2697 v4 18 cores, a 2.3 GHz
SSD per node	110 GiB
RAM per node	125 GiB
Network	100 Gb Omni-Path
Operating system	Ubuntu 20.04.5 LTS
MPI distribution	MPICH 4.1.1
GCC version	9.4.0
SLURM version	24.05.2

1.2. Expand Results

The following subsections will present the results of the evaluations performed using the benchmarks, described in Deliverable 3.1, with Expand on the HPC4AI Laboratory supercomputing cluster.

1.2.1. IOR Benhmark

We conducted performance tests to assess the Expand with IOR and measure the throughput for simultaneous read and write access on the same file and independent files per process, comparing its results with those obtained using the BeeGFS parallel file system, which is the one used in the HPC environment as a backend. All tests have been repeated 10 times, and for a 95% confidence interval, the error is less than 5% for all values.

The following configurations have been used for testing with IOR:

- Compute nodes: 1, 2, 4, 8, 16, 32, and 64.
- File systems: BeeGFS, Expand, and GekkoFS.
  - Local storage device for Expand file systems: SSD and shared memory (SHM) as underlying storage (`/dev/shm`).
- IOR workload:
  - Transfer size: 64 KiB, 512 KiB, and 1,024 KiB.
  - Client processes per compute node: 8.
  - Operations: write and read in parallel on a shared file and one file per process.
  - Size written by each client: 4 GiB (resulting in a 2 TiB of shared file in the maximum configuration).
  - Number of IOR iterations executed: 10 times.

Figures 1.2.1 and 1.2.2 show the results obtained in the evaluation carried out using the IOR benchmark with a shared file among all the processes and an individual file per process, respectively, on the HPC4AI Laboratory supercomputing cluster. The graphs show the throughput (in MiB/second in logarithmic scale) when performing write and read operations using transfer sizes of 64 KiB, 512 KiB, and 1024 KiB on Expand with SSD (XPN-SSD), and shared memory (XPN-SHM), and BeeGFS.

Figure 1.2.1 shows the results when using a shared file between all processes. By analyzing the performance obtained in write operations, we can see that the throughput in Expand (XPN-SSD and XPN-SHM) improves when the number of nodes is increased, scaling with the number of nodes. In the case of BeeGFS, from 4 nodes onwards, the performance is maintained.

When the transfer size is 64 KiB, the write performance of Expand is higher than BeeGFS in all cases. However, when the transfer size is 512 KiB and 1024 KiB, the performance delivered by BeeGFS is slightly higher than that of Expand with a few compute nodes. Even more, since Expand scales better, Expand's performance is higher for 8 nodes and above.

On the other hand, if we analyze the read operations, Expand performs better than the BeeGFS file systems, regardless of the device used for local storage. For example, when using a transfer size of 64 KiB and 32 compute nodes, Expand offers 2 orders of magnitude performance over BeeGFS. Moreover, the throughput increases as the number of compute nodes increases.

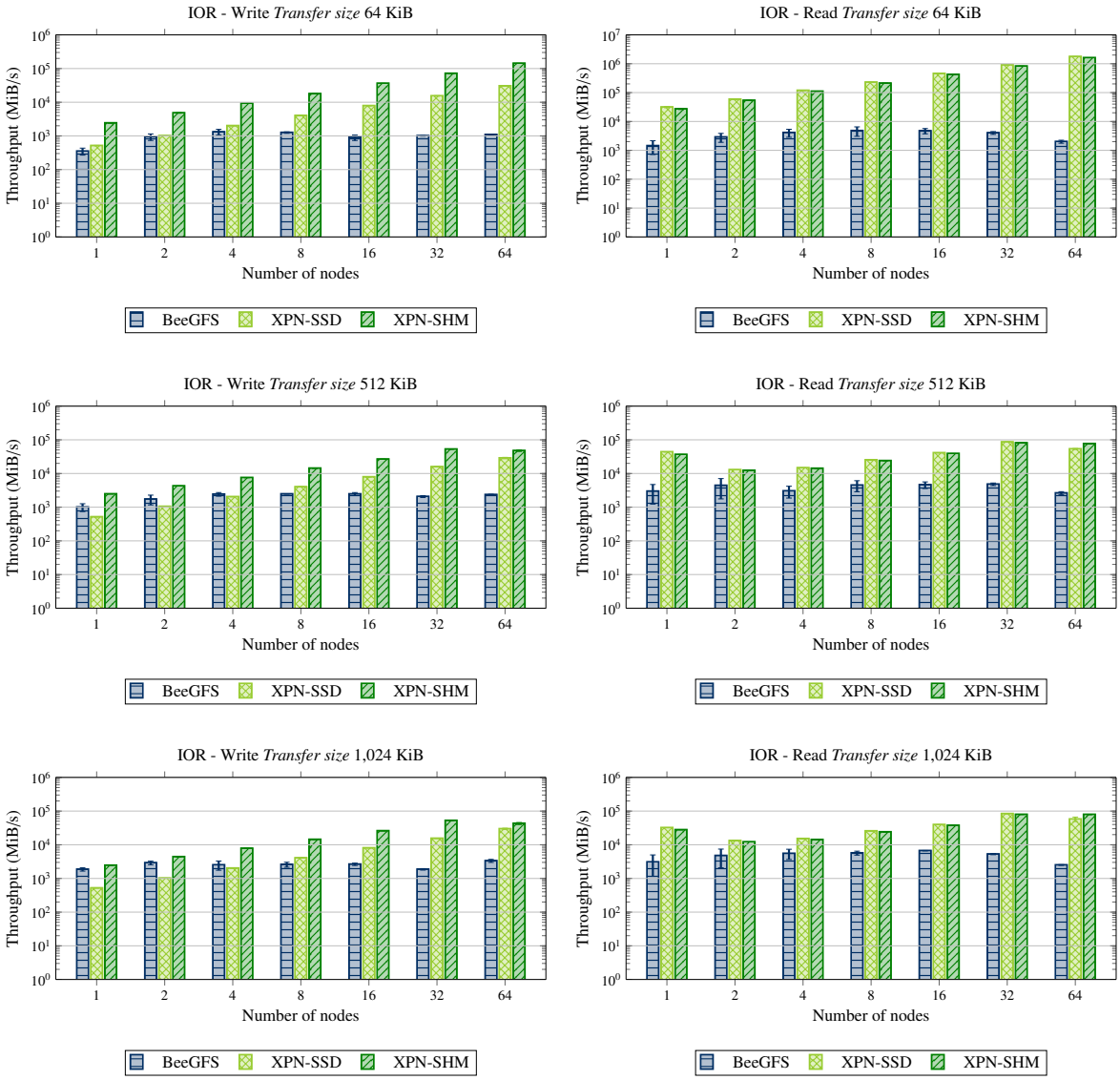

Figure 1.2.1: HPC4AI Laboratory: BeeGFS vs. Expand. Throughput (MiB/s) writing and reading data with different transfer sizes (64 KiB, 512 KiB, and 1,024 KiB), compute nodes (1, 2, 4, 8, 16, 32, and 64), with 8 client processes per node and **shared file**. Results in **logarithmic scale**.

Regarding the results obtained when using one file per process, we can see in Figure 1.2.2 that in the case of BeeGFS, the throughput in writing hardly changes when the number of nodes increases. It does not scale as the number of nodes increases. As with file sharing, Expand increases performance along with the number of nodes.

The BeeGFS file system offers better write performance than Expand when using up to 8 compute nodes. However, in evaluations performed with more than 8 nodes, Expand provides higher throughput.

If we study the behavior of these file systems when read operations are performed, we can see that Expand obtains a higher throughput than BeeGFS, as is the case when shared files are used. Moreover, it is worth noting that as the number of compute nodes increases, the Expand throughput increases, contrary to BeeGFS, which remains almost constant.

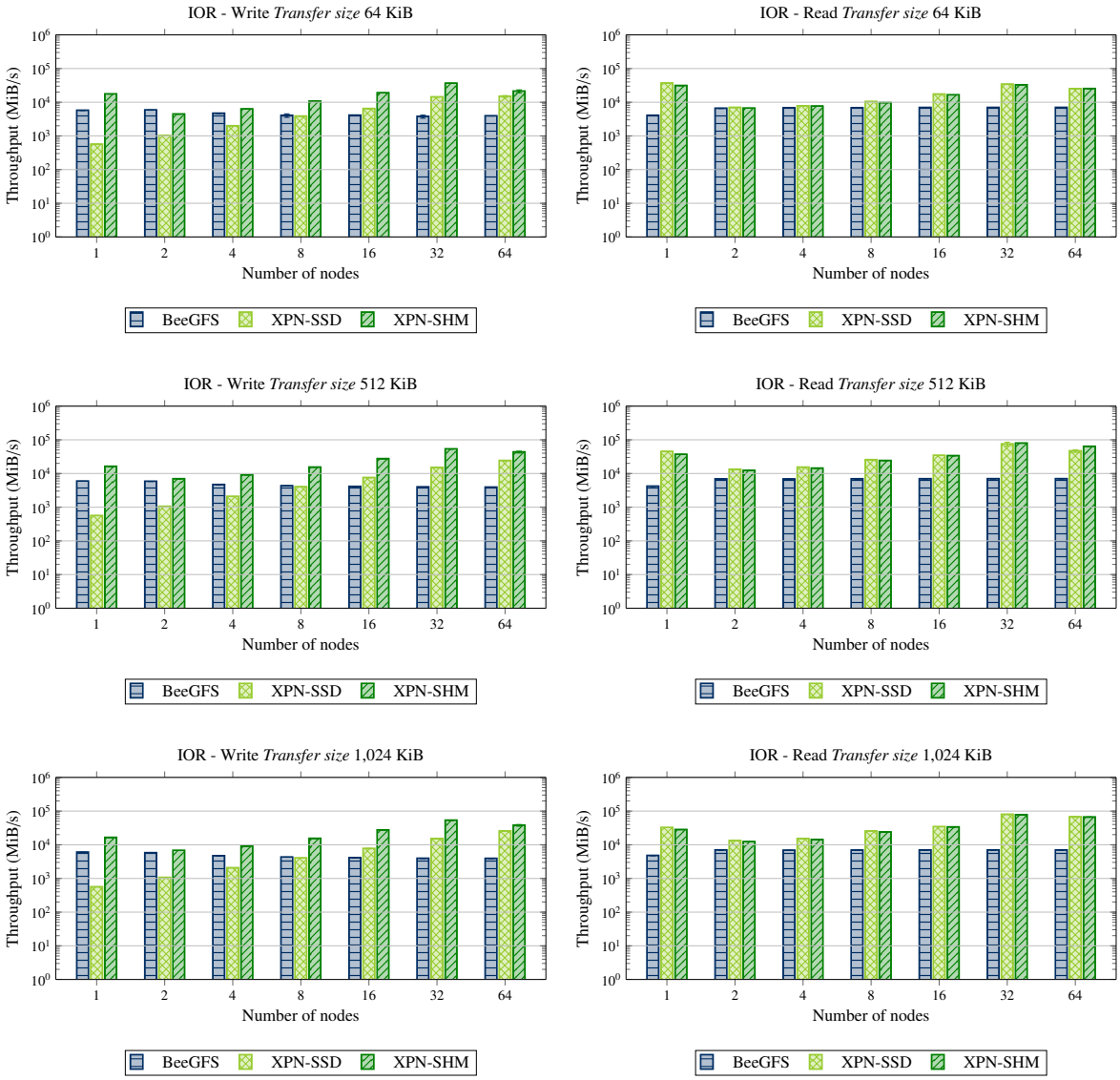

Figure 1.2.2: HPC4AI Laboratory: BeeGFS vs. Expand. Throughput (MiB/s) writing and reading data with different transfer sizes (64 KiB, 512 KiB, and 1,024 KiB), compute nodes (1, 2, 4, 8, 16, 32, and 64), with 8 client processes per node and **file per process**. Results in **logarithmic scale**.

1.2.2. DLIO Benhmark

To evaluate the performance of Expand with Deep Learning applications, it has also been evaluated with the DLIO benchmark in the HPC4AI Laboratory supercomputing cluster, comparing its results with those obtained using the BeeGFS parallel file system, which is the one used in the HPC environment as a backend. The DLIO configurations used for these evaluations are:

- Compute nodes: 16, 32, and 64.
- File systems: BeeGFS and Expand.
  - Local storage device for Expand file system: SSD and shared memory (SHM) as underlying storage (`/dev/shm`).
- DLIO workload:
  - BERT: Dataset size: 373 GiB. epochs: 1.
  - CosmoFlow: Dataset size: 65 GiB. epochs: 4.
  - ResNet50: Dataset size: 147 GiB. epochs: 1.
  - UNET3D: Dataset size: 36 GiB. epochs: 10.

Both file systems' I/O throughput (MiB/seconds) has also been evaluated when running the workload offered by the DLIO benchmark on the HPC4AI Laboratory supercomputing cluster. The results obtained on BeeGFS, Expand using SSD (XPN-SSD), and shared memory (XPN-SHM) in such evaluation can be seen in Figure 1.2.3.

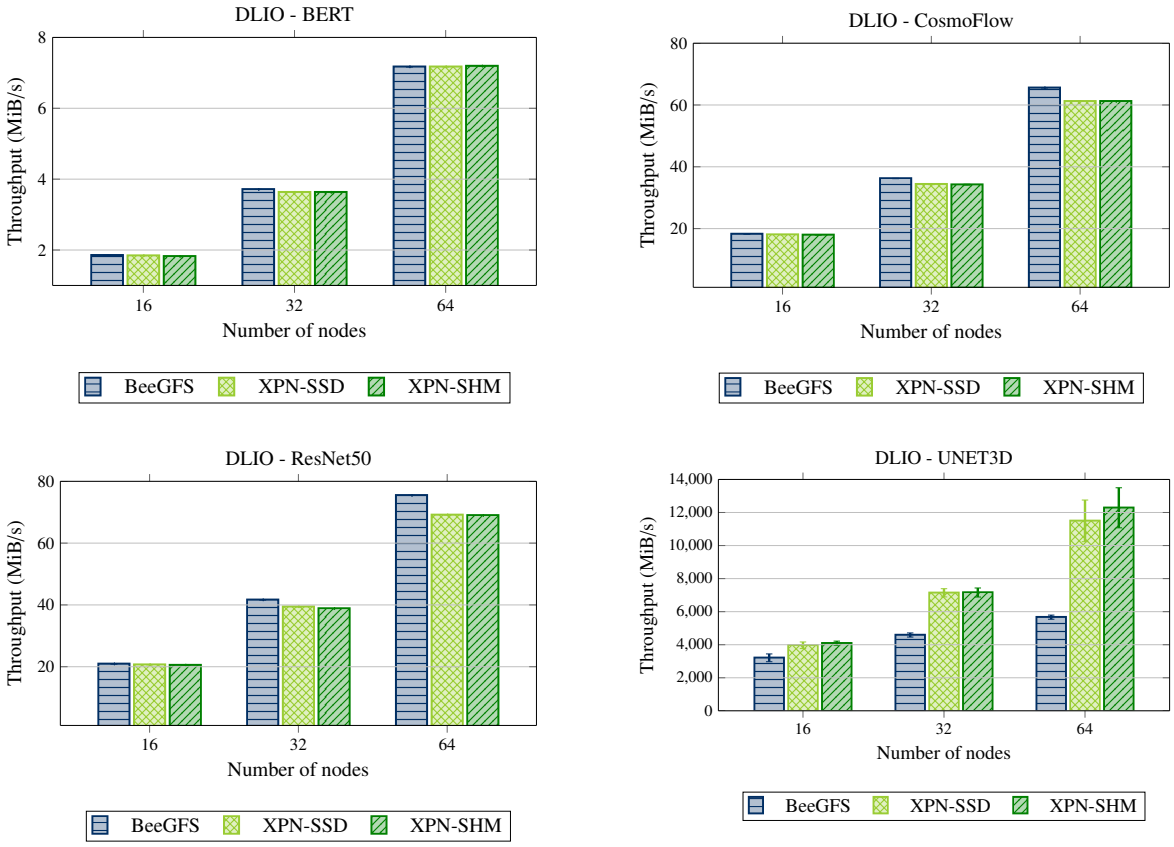

Figure 1.2.3: HPC4AI Laboratory: BeeGFS vs. Expand. Throughput (MiB/s) of DLIO training with different workloads (BERT, ComsoFlow, ResNet50, and UNET3D) and compute nodes (16, 32, and 64).

After running the different DLIO workloads, we can see that Expand gives similar results to BeeGFS for BERT, CosmoFlow, and ResNet50, even though the number of epochs are 1, 4, and 1, respectively. Although the

results for 32 and 64 nodes for BeeGFS are higher, the difference in the worst case, ResNet50, is lower than around 6%. In the case of UNET3D working with 10 epochs, the throughput is higher for Expand in all cases (up to 56% more).

As can be seen from the results obtained with DLIO, the number of epochs plays a crucial role in achieving these results. When the number of epochs increases, Expand takes advantage of the locality of the data when using local storage and becomes more efficient while avoiding remote access as the data is stored locally in the compute nodes that run the training, as seen in the UNET3D use case. In the case of UNET3D, 10 epochs are performed on the same dataset, while CosmoFlow, BERT, and ResNet50, 4, 1, and 1 epochs are performed, respectively.

On the other hand, when only one epoch is performed, the difference in remote access is less noticeable since the data locality is not fully exploited, as seen in the BERT and ResNet50 use cases. Moreover, Expand has to preload the initial data set, while BeeGFS does not. All these factors mean that the results obtained in Expand are not better than expected. However, in real training applications, the number of epochs will be higher than one epoch because a higher number means better training. With a higher number of epochs, the result obtained for Expand will be much better, as seen in the evaluations, since the data for training will already be available in the compute nodes used by the application.

1.2.3. IO500 Benchmark

Expand has also been evaluated with the IO500 benchmark in different HPC environments. Specifically, it has been evaluated on MareNostrum 4 [2], Leonardo [3], and C3-UC3M. These evaluations have allowed to know the global performance of the file system and to include Expand in the IO500 list, where the following positions are as follows:

- MareNostrum 4 supercomputer evaluation (SC23):
  - 69th position out of 101 in the 10 Node Research list.
  - 184th position out of 236 in the Full list.
- Leonardo supercomputer evaluation (SC24):
  - 47th position out of 113 in the 10 Node Research list.
  - 114th position out of 268 in the Full list.
- C3-UC3M supercomputer evaluation (ISC25):
  - 78th position out of 118 in the 10 Node Research list.
  - 205th position out of 284 in the Full list.

1.3. Expand with Fault Tolerant results

In the evaluation figures, the following acronyms will be used and can be found in Table 1.3.1.

Table 1.3.1: Acronyms in the evaluation figures.

Acronym	Definition
BeeGFS	BeeGFS backend file system results.
XPN	Expand Ad-Hoc results by default.
XPN-Repl-2	Expand Ad-Hoc results with replication factor 2.
XPN-Err-1	Expand Ad-Hoc results with replication factor 2, where 1 of them is a server with an error.
XPN-Repl-4	Expand Ad-Hoc results with replication factor 4.
XPN-Err-3	Extend Ad-Hoc results with replication factor 4, where 3 of them are servers with an error.

1.3.1. IOR

In order to evaluate the fault tolerance in Expand Ad-Hoc, the IOR benchmark was initially utilized to assess the bandwidth of a parallel application accessing a shared file. The objective of this evaluation is to determine the impact of data replication and the impact of having servers with errors.

The evaluation was also conducted in BeeGFS to allow for a comparison with the supercomputer's backend parallel file system. Additionally, the evaluation was conducted in Expand Ad-Hoc without replication and with a replication factor of 2 and 4. In addition, we also run the evaluation with one server exhibiting an error with Expand Ad-Hoc with a replication factor of 2 and three servers exhibiting an error with Expand Ad-Hoc with a replication factor of 4. To simulate this error, the servers were terminated at the start of the benchmark.

In the evaluation, 8 client processes are utilized per node. Each client process is responsible for reading and writing 1 GiB of data, resulting in a maximum of 512 GiB in the greater configuration with 64 nodes. Accordingly, the results of this evaluation are presented in Figure 1.3.1.

As illustrated in Figure 1.3.1, the superiority of Expand Ad-Hoc in its default configuration is evident when compared to BeeGFS in all the tests.

Regarding the impact of data replication, the replication factor affects the performance of write operations. This is because as the replication factor increases, the volume of data to be written also increases, resulting in a corresponding decrease in performance. However, concerning the read operation, the performance improves in proportion to the replication factor due to the increased locality of the data and the optimizations previously discussed.

The results obtained with the servers exhibiting errors indicate that the impact on the write operation is minimal. This comparison is made between the servers without error and with error, being respectively (XPN-Repl-2 and XPN-Err-1) and (XPN-Repl-4 and XPN-Err-3). This is because the client effectively avoids the server with the error, thereby maintaining normal functionality. In contrast, the read operation exhibits a notable decline in performance, which increases with the number of servers with errors. This is because, in the event of an error in the server, the client is forced to redirect its access to the remaining servers that possess the necessary data. Consequently, when a considerable number of clients are involved, the servers are subjected to excessive loads, resulting in a notable decline in performance, as evidenced by the results.

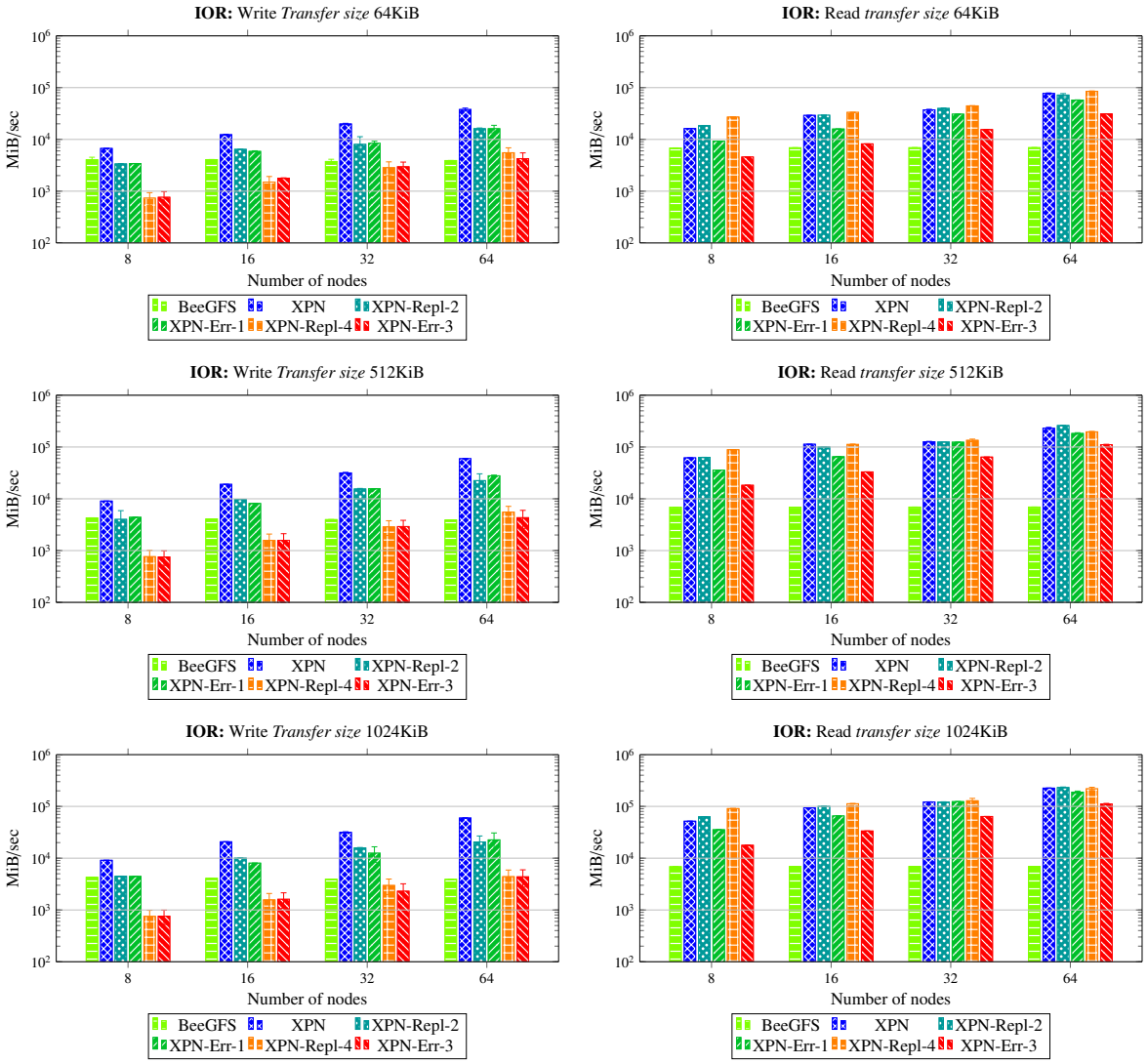

Figure 1.3.1: BeeGFS vs. Expand Ad-Hoc with fault tolerance. Bandwidth (MiB/sec) writing and reading with different replication factors (1 (XPN), 2 (XPN-Repl-2), and 4 (XPN-Repl-4)), error in servers (1 (XPN-Err-1) and 3 (XPN-Err-3)) transfer sizes (64KiB, 512KiB, and 1MiB), and compute nodes (8, 16, 32, and 64), with 8 client processes per node and **file share**. Results in logarithmic scale.

It is also worth noting that the performance of the smallest transfer size, 64 KiB, is relatively inferior to that of the other transfer sizes. This is because a smaller transfer size results in a greater number of operations, which ultimately leads to a decline in performance. However, the performance remains relatively consistent with the other two transfer sizes.

In conclusion, the Expand Ad-Hoc parallel file system exhibits superior performance in read operations compared to the BeeGFS parallel file system despite the presence of data replication and potential errors in the underlying servers. In the write operation, the default configuration and up to replication factor 2 are also superior, with replication factor 4 only exceeding BeeGFS in some configurations.

1.3.2. DLIO

The following evaluation uses the DLIO benchmark to assess the bandwidth performance in a deep learning environment. In the DLIO, the UNET3D workload, consisting of a 3D medical image segmentation with a dataset size of 36 GiB and run for 10 epochs, is selected. The results of this evaluation are presented in Figure 1.3.2.

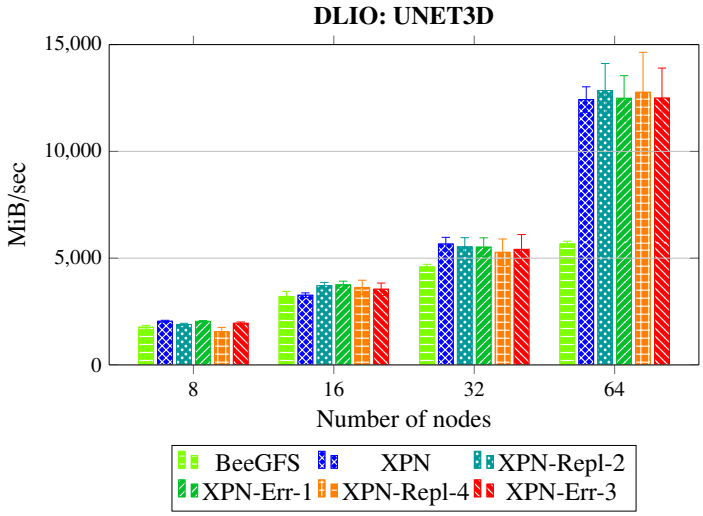

Figure 1.3.2: BeeGFS vs. Expand Ad-Hoc with fault tolerance and failing servers. Bandwidth (MiB/sec) of training performed by DLIO with 8, 16, 32, and 64 compute nodes.

As illustrated in Figure 1.3.2, in this particular scenario, the replication factor and the servers with errors have a negligible impact on the performance of Expand Ad-Hoc. This is because this deep learning benchmark is centered around read operations without overloading the servers. Additionally, Expand Ad-Hoc demonstrates superior performance in all configurations compared to BeeGFS. Notably, with 64 compute nodes, Expand Ad-Hoc outperforms BeeGFS, which performs at approximately double its bandwidth. This is because BeeGFS is the backend file system of the supercomputer and is not scalable with an increase in nodes. Once it reaches its maximum performance, it is unable to scale further. In contrast, the Expand Ad-Hoc file system is an ad-hoc parallel file system capable of scaling its performance with the compute nodes due to the deployment of servers within the nodes.

1.4. Expand with Malleability results

1.4.1. Without replication

The objective of this experiment is to assess the overhead that is generated when a malleability operation is performed. For this purpose, server startup and shutdown times and the time it takes to move data from one partition to the new partition will be measured.

To achieve this, a test was developed whereby a file of 16 GiB of data was initially generated using the IOR benchmark. Subsequently, the malleability operation is executed, and the corresponding elapsed times are

recorded. Finally, the IOR benchmark is employed to validate the data's integrity, ensuring no corruption occurs during the malleability operation.

This is achieved through the utilization of a specific feature present within the IOR benchmark. An internal hash for the data within the file is employed when a timestamp is provided to the IOR in a write operation. This hash is then utilized internally in a subsequent read operation to verify the integrity of the data. This test allows us to verify the correct execution of these malleability operations on the data while measuring the overhead. The results of this evaluation are presented in Figure 1.4.1.

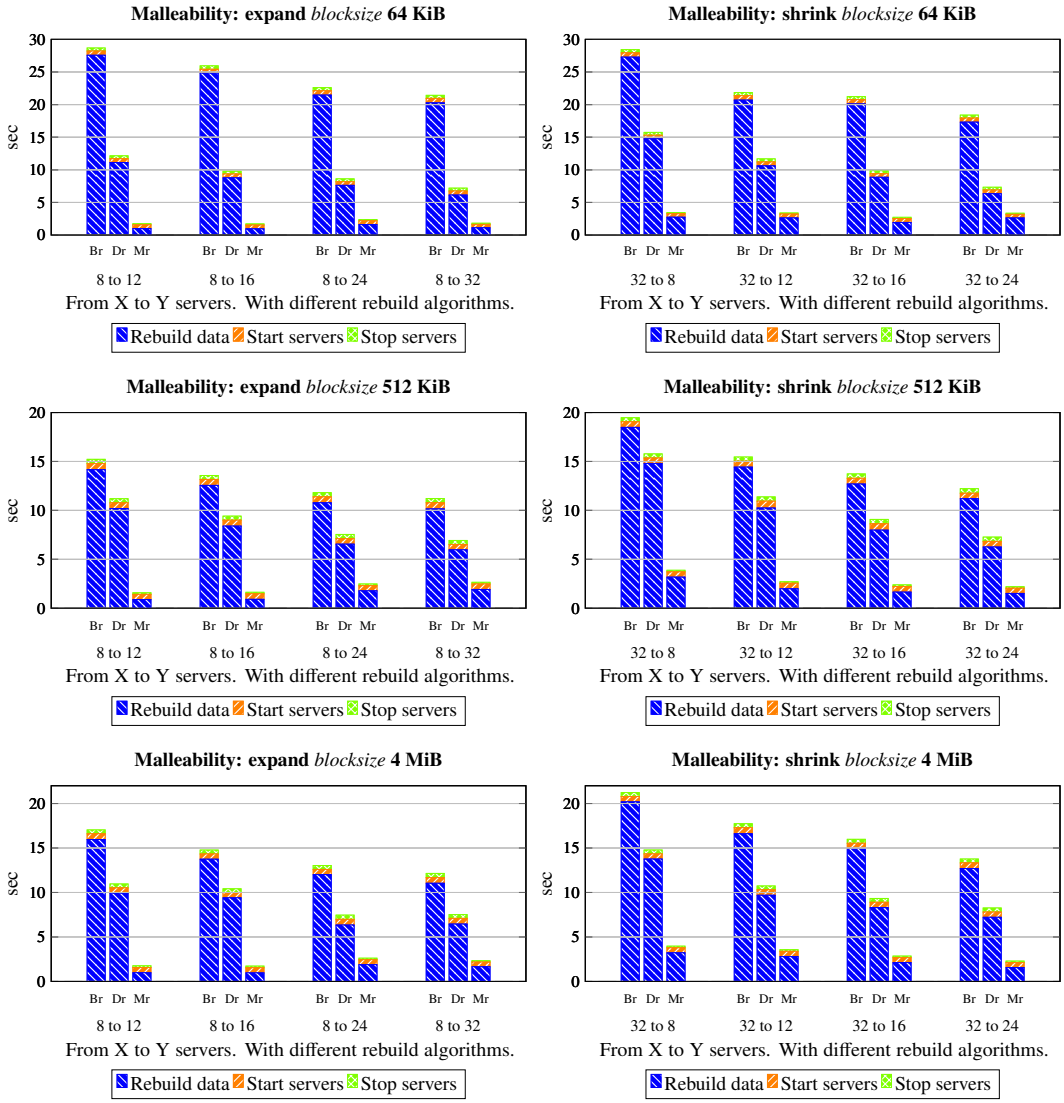

Figure 1.4.1: Rebuild in Expand Ad-Hoc with malleability. It shows the seconds to rebuild the data, stop the old servers, and start the new ones in Expand Ad-Hoc containing 16 GiB of data with different numbers of servers, different transfer sizes (64 KiB, 512 KiB, and 4 MiB), compute nodes (8, 12, 16, 24, and 32). Also, with different versions of the *Br* (backend rebuild), *Dr* (direct rebuild), and *Mr* (metadata rebuild) algorithms.

As illustrated in Figure 1.4.1, the results showcase the efficacy of the diverse malleability algorithms presented. As previously stated, the backend rebuild (**Br**) is the less efficient of the two. This is because it utilizes two operations to transfer the data and the use of the supercomputer's backend file system.

Therefore, the backend rebuild algorithm (**Br**) requires the greatest amount of time to transfer the data. It is worth mentioning that when the block size is particularly small, as evidenced by the results presented with 64 KiB, the performance of this algorithm tends to decline in comparison to other block sizes.

The direct rebuild (**Dr**) is a single-operation process that performs a complete reconstruction of the file. Compared with the backend rebuild, the direct rebuild is a significantly more efficient method, exhibiting an average 2.7 times increment in performance with the block size of 64 KiB. In addition, the remaining file sizes observed an average speedup of 1.7.

The metadata rebuild (**Mr**) algorithm is the most efficient of those considered, exhibiting an average speedup of 4.7 in all operations when compared to the direct rebuild. This is due to the approach of this algorithm, which involves moving only the necessary data and updating the metadata of the file.

1.4.2. With replication

This next test aims to verify the correct functionality and implementation of the data replication within the malleability operations. To this end, the most suitable algorithm, specifically the metadata rebuild (**Mr**), will be evaluated at different replication factors. The results can be seen in Figure 1.4.2.

As anticipated and in accordance with the cost functions previously calculated, the findings illustrated in Figure 1.4.2 align with the expected outcome.

The expansion operation exhibits consistent time duration across all replication factors, as there is no data movement, only metadata updates. This is attributed to the consistent number of files across all replication factors.

Similarly, the shrink operation demonstrates a direct correlation with the replication factor, exhibiting increased time duration with higher replication factors. This is attributed to the transfer of a greater volume of data with greater replication factors.

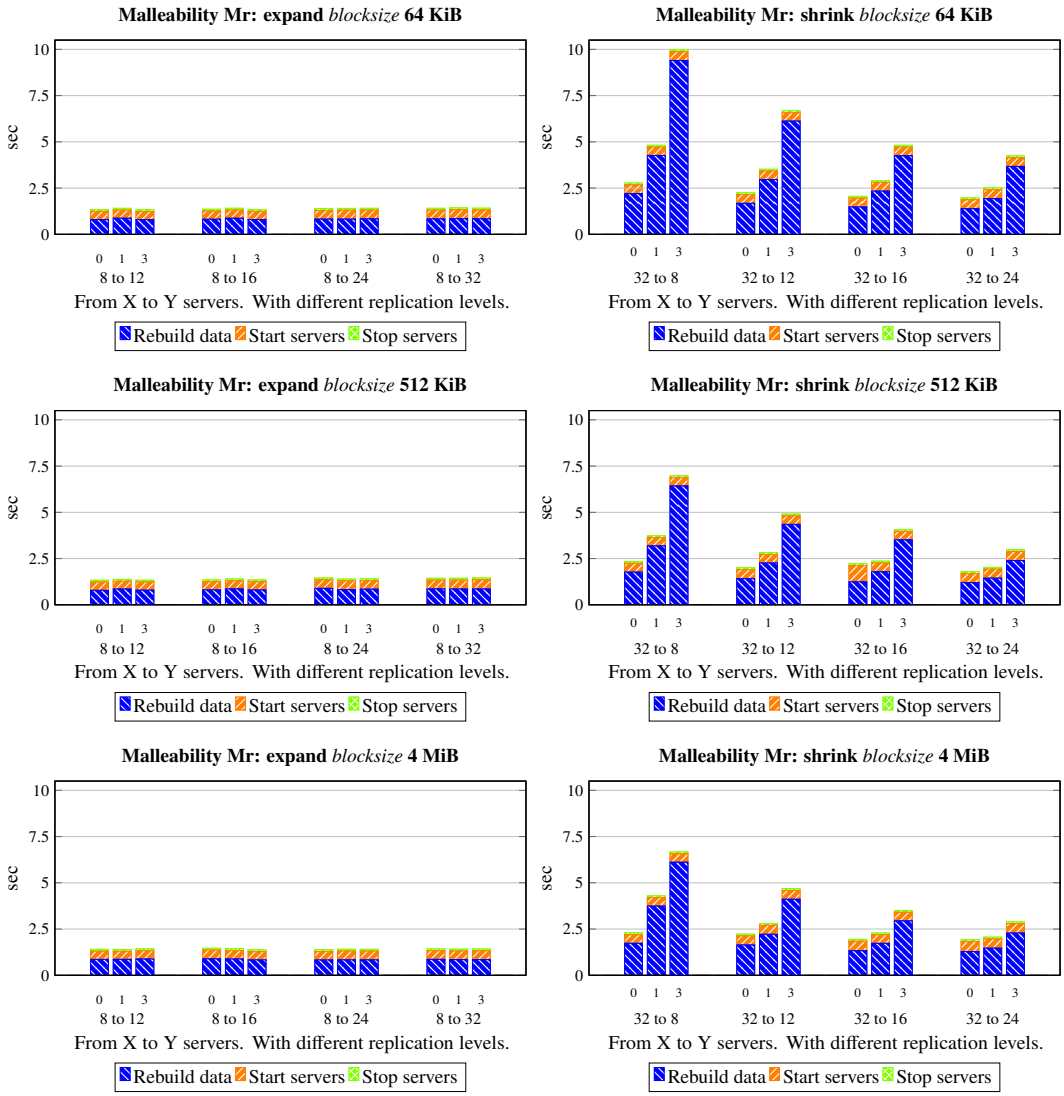

Figure 1.4.2: Rebuild in Expand Ad-Hoc with metadata rebuild (Mr) malleability algorithm and replication factors (0, 1, and 3). It shows the seconds to rebuild the data, stop the old servers, and start the new ones in Expand Ad-Hoc containing 16 GiB of data with different numbers of servers and compute nodes (8, 12, 16, 24, and 32) and different transfer sizes (64 KiB, 512 KiB, and 4 MiB).

2. RESULTS FOR REAL APPLICATIONS

This chapter presents the results of Expand evaluations using different real applications. The results when Expand does not use fault tolerance and malleability will be shown in Section 2.1, the results when using fault tolerance will be presented in Section 2.2, and the results when using malleability will be introduced in Section 2.3.

2.1. Expand Results

The following subsections will present the results of the evaluations performed using the real applications, described in Deliverable 3.1, with Expand on the HPC4AI Laboratory supercomputing cluster.

2.1.1. EpiGraph

In this subsection, the execution time of EpiGraph and the bandwidth of the I/O operations performed by this application will be evaluated. For this purpose, the following configurations will be used:

- Compute nodes: 4, 8, 16, 32, and 64.
- File systems: BeeGFS and Expand.
  - Local storage device for Expand file system: SSD.
- EpiGraph workload:
  - Simulated days: 130.
  - EpiGraph processes per node: 20.
  - Checkpoint files: single and multiple.

Figure 2.1.1 shows the evaluation results performed with EpiGraph using different numbers of compute nodes and configurations of the checkpoint file. In particular, one configuration of the checkpoint file consists of keeping a single checkpoint file, i.e., this file is overwritten, maintaining only the last state of the simulation. The other type of configuration consists of maintaining a file with each of the generated checkpoints, storing all the states of the simulation. The bar chart shows the execution time of the application, in seconds, and the line chart shows the bandwidth of the I/O operations, in MiB/s.

On the one hand, if we study the total execution time of EpiGraph, it can be seen that the execution time when using Expand as the file system is less than when using BeeGFS. Especially when using 64 compute nodes, the execution time with Expand is less than half that with BeeGFS. Furthermore, it can also be seen that with both checkpoint file configurations and both file systems, the total execution time decreases progressively up to 32 compute nodes, but increases substantially with 64 nodes. This behavior is because 1,280 EpiGraph client

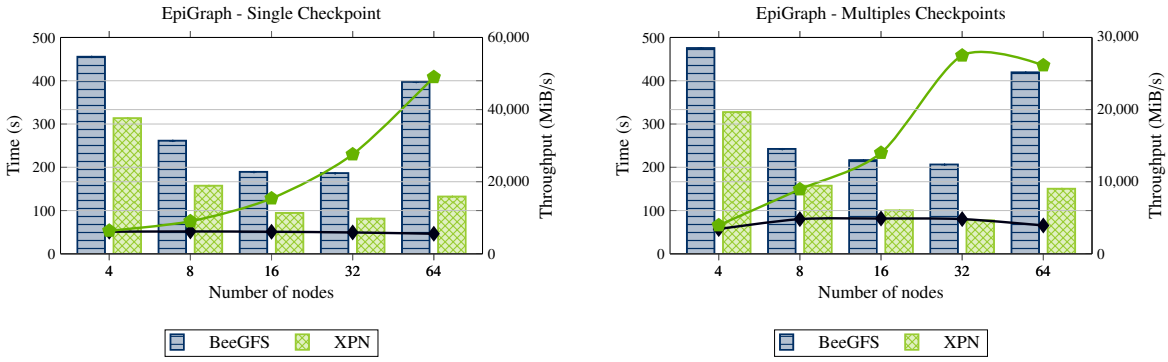

Figure 2.1.1: HPC4AI Laboratory: BeeGFS vs. Expand with EpiGraph. In columns, the execution time (seconds), and in lines, the bandwidth (MiB/s) of I/O operations performed by the Epigraph real-world application with different checkpoint configurations and compute nodes (4, 8, 16, 32, and 64).

processes are running in parallel when using 64 compute nodes, and the synchronization of these processes is more expensive than the computation they each have to perform, which causes the execution time to increase.

On the other hand, if we analyze the bandwidth of the I/O operations performed by EpiGraph, it can be seen that when BeeGFS is used as the file system, it remains constant, regardless of the number of compute nodes. However, when using Expand, the bandwidth scales with the number of compute nodes, except when using 64 nodes and multiple checkpoint files, it stabilizes.

2.1.2. Nek5000

In this subsection, we will study the execution time of Nek5000 and the bandwidth of the I/O operations it performs. For this purpose, the following configurations will be used:

- Compute nodes: 16, 32, and 64.
- File systems: BeeGFS and Expand.
  - Local storage device for Expand file system: SSD.
- Nek5000 workload:
  - Use case: TurbPipe.
  - Total steps: 10,000.
  - Nek5000 processes per node: 32.
  - Output writing interval (steps): 1, 2, 5, and 10.
  - I/O interface: MPI-IO.

Figure 2.1.2 shows the evaluation results performed with Nek5000 with different write intervals (writeInterval) and number of compute nodes. The bar chart shows the total execution time of the application, in seconds, and the line chart shows the bandwidth of the I/O operations, in MiB/s.

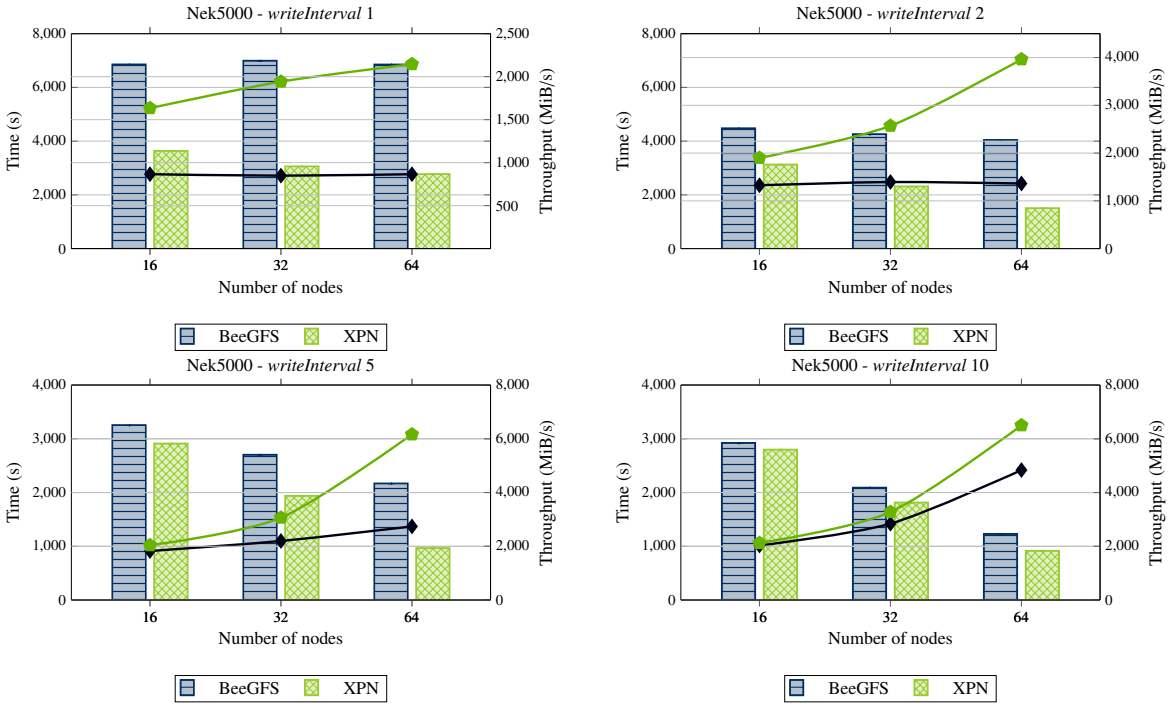

Figure 2.1.2: HPC4AI Laboratory: BeeGFS vs. Expand with Nek5000 TurbPipe use case. In columns, the execution time (seconds), and in lines, the bandwidth (MiB/s) of I/O operations performed by the Nek5000 real-world application with different write intervals (1, 2, 5, and 10) and compute nodes (16, 32, and 64).

On the one hand, if the total execution time of the application is analyzed, it can be seen that the total time decreases as more compute nodes are used, regardless of the write interval and the file system. This is because there are more client processes and more parallelism. However, it can be seen that the execution time of Nek5000 is lower when Expand is used as the file system in all cases, especially when the write intervals are smaller. Mainly, when writing in all steps, the execution time with Expand becomes less than half that with BeeGFS.

On the other hand, if we study the bandwidth of I/O operations, it can be seen that in the case of Expand, it always scales with the number of compute nodes, regardless of the write interval. In the case of BeeGFS, it can only scale when this interval is increased to 5 and 10 steps.

This behavior is mainly due to two reasons. First, Expand uses the local storage of the compute nodes, which allows it to exploit the locality of the data and reduce the number of remote accesses. Second, since MPI-IO is used as the I/O interface on Nek5000, the resulting files are shared among all processes, so BeeGFS has to apply consistency mechanisms that substantially compromise I/O performance.

2.1.3. Remote Sensing

In this subsection, the execution time and bandwidth of the I/O operations performed will be analyzed, as in the previous applications. For this purpose, the following configurations will be used:

- Compute nodes: 4, 8, 16, 32, and 64.
- File systems: BeeGFS and Expand.
  - Local storage device for Expand file system: SSD.
- Remote Sensing workload:
  - Epochs: 500 epochs.
  - Data size: 1.12 GiB for training, 0.55 GiB for validation, and 0.55 GiB for test.

Figure 2.1.3 shows the evaluation results performed with Remote Sensing using a different number of compute nodes. The bar chart shows the total execution time of the application, in seconds, and the line chart shows the bandwidth of the I/O operations, in MiB/s.

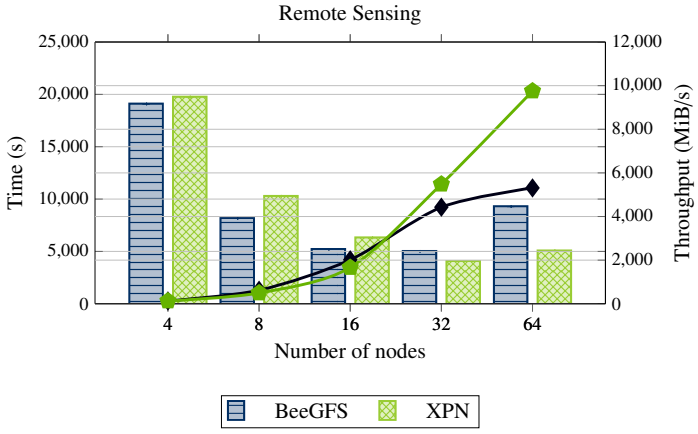

Figure 2.1.3: HPC4AI Laboratory: BeeGFS vs. Expand with Remote Sensing. In columns, the execution time (seconds), and in lines, the bandwidth (MiB/s) of training performed by the Remote Sensing Deep Learning real-world application with different compute nodes (4, 8, 16, 32, and 64).

On the one hand, regarding the execution time of this application, it can be seen that it is lower when using the BeeGFS file system than when using Expand with 4, 8, and 16 compute nodes. However, this behavior changes when 32 and 64 computation nodes are used, where Expand achieves a lower execution time with respect to BeeGFS, which is approximately half when 64 computation nodes are used.

On the other hand, regarding the bandwidth of I/O operations, it can be seen that Expand scales with the number of nodes, while BeeGFS bandwidth stabilizes from 32 compute nodes onwards.

This behavior is because Expand, an ad-hoc file system, increases the number of ad-hoc servers with the real application. BeeGFS has a fixed number of storage nodes that are also shared by all the applications in the HPC environment.

2.2. Expand with Fault Tolerant results

2.2.1. Remote sensing

In order to complete the assessment of the fault tolerance design in Expand Ad-Hoc, we employ a real-world application. This application employs Horovod to train a multispectral (comprising not only RGB channels but also infrared) ResNet convolutional neural network (CNN) on BigEarthNet, a large remote sensing dataset, while performing a classification on a subset of the dataset. The classification problem is multi-label, meaning that more than one label can be associated with each sample [4].

Due to the design's nature, the total amount of data read is directly proportional to the total number of processes involved. Consequently, a lower number of processes results in less data being read. The application utilizes a dataset size of 1.19 GiB for training and 0.55 GiB for validation and testing and runs for 500 epochs. The results of this evaluation are presented in Figure 2.2.1. This evaluation considers the time required to transfer data from the backend file system to Expand Ad-Hoc. Despite the necessity of data transfer, the use of Expand Ad-Hoc has shown better results compared to the backend file system, as evidenced by the findings of this evaluation.

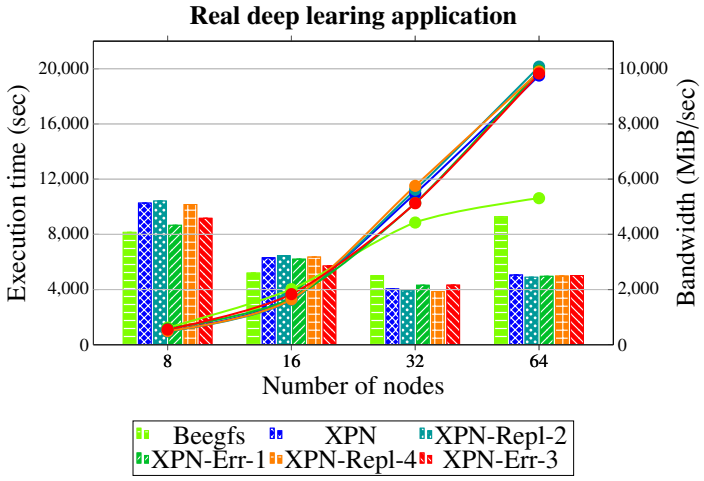

Figure 2.2.1: BeeGFS vs. Expand Ad-Hoc with fault tolerance. In columns, the time of the execution (sec) which takes into account the time it takes to initially load the data to Expand Ad-Hoc, and in lines, the training bandwidth (MiB/sec) performed by the real deep learning application with 8, 16, 32, and 64 compute nodes.

As illustrated in Figure 2.2.1, the results are similar to those obtained when the DLIO benchmark was evaluated with fault tolerance, particularly with regard to the bandwidth obtained in the application, which is represented by the lines. As with the previous benchmark, the Expand Ad-Hoc solution demonstrates superior performance from 16 nodes onwards. It is noteworthy that, as in the previous evaluation, the configuration with 64 compute nodes produces approximately double the bandwidth, which translates to approximately half the execution time compared to BeeGFS. Similarly, as with the previous benchmark, the replication factor and the servers with errors have a negligible impact on performance, given that the application is a deep learning training centered on read operations. This, combined with the fact that the application does not saturate the server, the data locality and the ad-hoc implementation that scales with the application allows Expand Ad-Hoc to achieve better performance than the BeeGFS backend parallel file system.

2.3. Expand with Malleability results

2.3.1. WaComM

To conclude the assessment of malleability, the anticipated outcomes are corroborated by using a real-world application, WaComM. The application's functionality is as follows: first, it reads an input file containing 2.62 GiB of data, then performs a particle simulation using that data, and finally writes the output to a file. This process is repeated for a multitude of files in a series.

In the designed test, the application is started with 100 processes that utilize 8 Expand Ad-Hoc servers. Upon the consumption of 4 files, an expand malleability operation is performed to expand from 8 to 16 Expand Ad-Hoc servers. Once the application has consumed an additional 4 files, a shrink malleability operation is performed to reduce the number of servers from 16 to 12. The application concluded with the consumption of an additional 4 files.

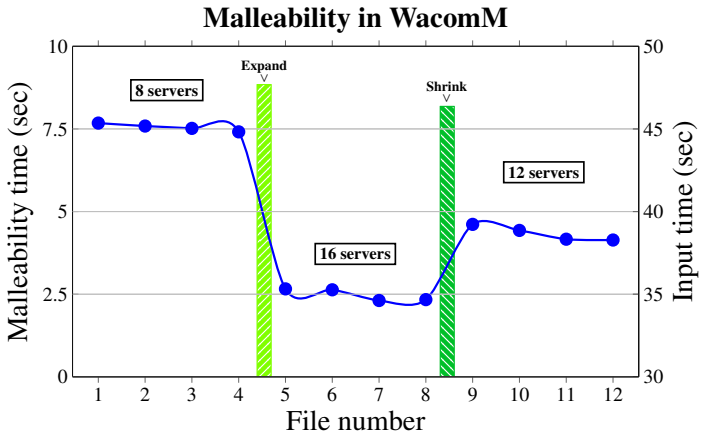

Figure 2.3.1: Expand Ad-Hoc with malleability in WacomM. In columns, the time of the execution of the malleability (sec), and in lines, the time taken to read the input file (sec). The number of servers begins with 8, then increases to 16, and later decreases to 12.

The results are presented in Figure 2.3.1, which can be interpreted as a timeline showing the progression of the files. The lines indicate the time required to read the input file, while the columns mark the time taken to perform a malleability operation. The malleability operation is the direct rebuild algorithm, which redistributes all the data.

As anticipated, the expand operation resulted in a reduction of the file reading time from 45 to 35 seconds, representing a 10-second saving per file and a cost of 8.8 seconds to perform the malleability. Subsequently, when the shrink operation is performed, the time required to read the file is increased from 35 to 38 seconds, with a cost of 8.2 seconds to perform the malleability. This is typically utilized when the application does not require the highest level of performance and when the objective is to conserve supercomputer resources.

Another method for demonstrating the operation of the application is to monitor the bandwidth of the servers and construct a time series with that data. This is illustrated in Figure 2.3.2, which presents a measurement taken every 500 milliseconds, illustrating the read bandwidth peaking at different times, representing the time the application is reading data from the servers. The write bandwidth also has peaks, representing when the

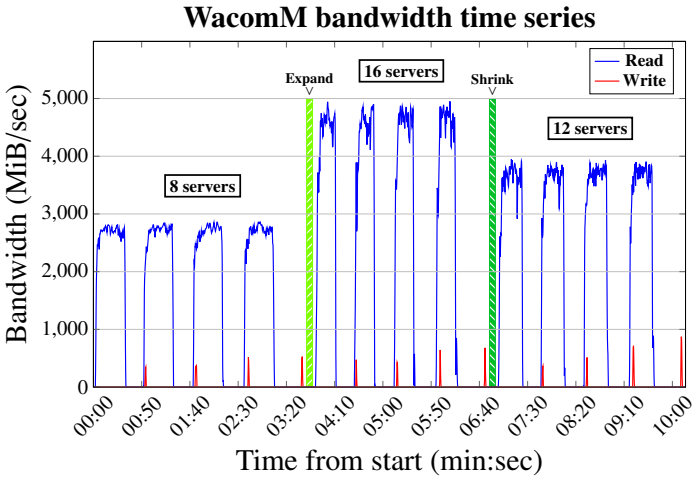

Figure 2.3.2: Expand Ad-Hoc with malleability in WacomM. The columns represent the malleability operations, and the lines indicate the bandwidth of the servers (MiB/sec) at each moment. The measurements are taken every 500 milliseconds. The number of servers begins with 8, then increases to 16, and later decreases to 12.

application saves its results to the file system in each cycle.

Additionally, as illustrated in Figure 2.3.2, the variation in bandwidth is evident when a malleability operation is conducted. As expected, an increase is observed when an expand malleability operation is performed, while a decrease is noted with a shrink malleability operation. This observation aligns with the earlier explanation of Figure 2.3.1.

3. RESULTS FOR BIG DATA BENCHMARKS

This chapter will describe the results obtained with the benchmarks used for the evaluation in Big Data environments. In Section 3.1, the environment used for this evaluation will be described. The results obtained for WordCount evaluation will be shown in Section 3.2 and the TeraSort ones will be presented in Section 3.3.

3.1. Environment description

The C3-UC3M supercomputer was used to evaluate Expand in Big Data environments. The characteristics of this environment are shown in Table 3.1.1.

Table 3.1.1: C3-UC3M supercomputer characteristics.

Attribute	**C3-UC3M**
Number of nodes	90
Nodes provider	Dell PowerEdge R6525
CPUs per node	2×64 AMD EPYC 7713
RAM per node	1 TiB DDR4
Network	Infiniband 100Gb/s
Operating System	Rocky Linux
MPI distribution	MPICH 4.3.0
GCC version	12.2.0
SLURM version	24.11.4
Spark version	3.5.1

3.2. WordCount

The WordCount evaluation was performed using 1 to 64 nodes with a file size of 2 TiB. The tests have been performed with a block size of 128 MiB for Expand. Since the block size of Lustre cannot be changed, the one configured on the supercomputer will be used.

The configuration used for WordCount is summarized as follows:

- Compute nodes: 1, 2, 4, 8, 16, 32, 64.
- File size: 2 TiB.
- File systems: Lustre and Expand.
  - Local storage device for Expand file system: NVMe as underlying storage (`/nvme`).
  - Block size for Expand file system: 128 MiB.

In Figure 3.2.1 can be seen the preload (1) and flush (2) execution times for Expand in C3-UC3M, both in linear scale. As can be seen, the preload time scales up to 2 nodes and after that, the scaling slows down. This is

due to the saturation of Lustre when performing I/O operations on shared files. This saturation is caused by the architecture of Lustre in this environment, since it has only two shared storage nodes among all compute nodes. In the case of the flush operation is not affected, since the application generates one file per process and this becomes easier for Lustre to handle.

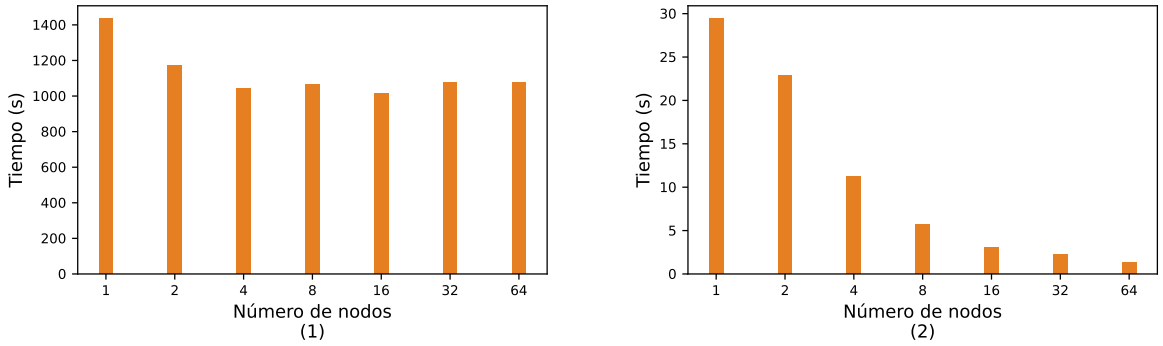

Figure 3.2.1: Expand preload (1) and flush (2) execution times in WordCount.

In Figure 3.2.2, the total execution times of the evaluation can be seen on a linear scale. The Lustre time shown is the execution of the application and the Expand time is the sum of the preload, execution and flush times. In this figure it can be seen how Lustre obtains lower times up to 2 nodes. From this number of nodes, Expand starts to obtain lower times given the rapid reduction of the execution time of the application. It can be seen how both systems scale up to 16 nodes and, from then on, execution times stabilize. However, it can also be seen how the main performance-limiting bottleneck in Expand executions is the saturation caused by Lustre in the preload operation.

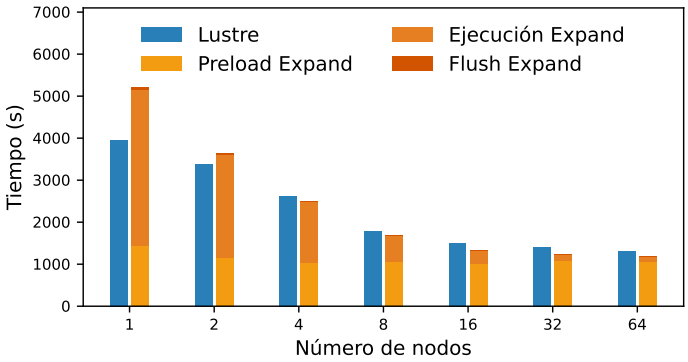

Figure 3.2.2: Lustre and Expand total execution times in WordCount.

Figure 3.2.3 shows the Speedup obtained during the evaluation. This Speedup has been obtained as follows:

$$S(N) = \frac{t(1)}{t(N)}$$

Where S(N) represents the Speedup for N nodes, N is the number of nodes and t(N) is the execution time measured for N nodes. In this way, it is possible to obtain the Speedup as a function of the resources used.

In this figure, it can be seen how the scaling is greater for Expand, but that both systems stop scaling after

16 nodes. However, since the bottleneck observed in Expand is due to Lustre saturation during the preload operation, Speedup has also been calculated without taking this time into account. In Figure **??**, this result is shown. As can be seen, for 64 nodes, Lustre triples the throughput using 64 times more resources, while Expand's throughput is multiplied by almost 35 times.

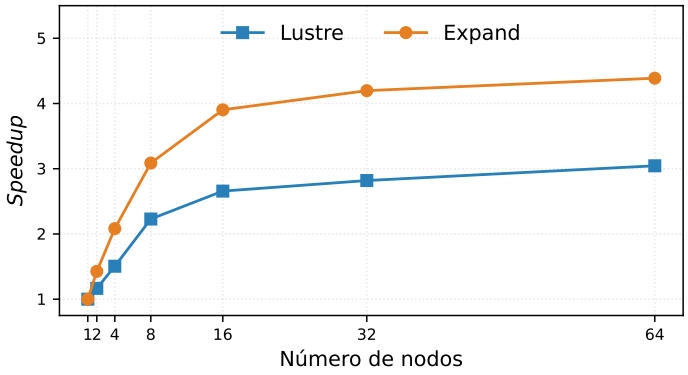

Figure 3.2.3: Lustre and Expand Speedup in WordCount.

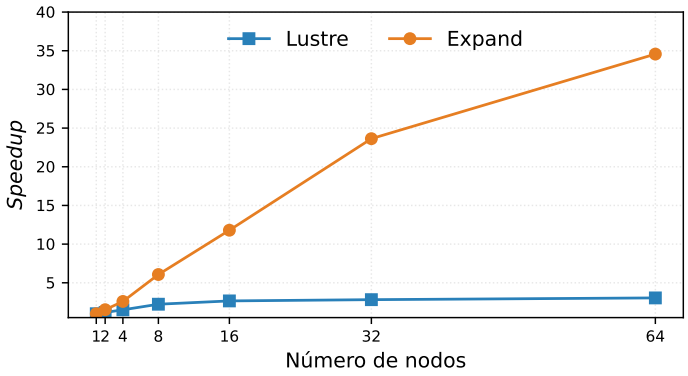

Figure 3.2.4: Lustre and Expand Speedup excluding preload time in WordCount.

3.3. TeraSort

For TeraSort, two tests were performed. First, a test with a large dataset (512 GiB) was executed. As in Wordcount, the block size selected for Expand 128 MiB. Secondly, a smaller test was performed. In this case, the block size selected was 8 MiB, and the file size was calculated proportionally to the previous evaluation, obtaining a file size of 32 GiB. This second test has been decided to be performed to compare the application's performance in both scenarios and verify its performance regardless of the file size used. For the 512 GiB file, from 2 to 64 nodes were used. On the other hand, for the 32 GiB file, from 1 to 64 nodes were used.

The configuration used for TeraSort is summarized as follows:

- File size: 512 GiB
  - Compute nodes: 2, 4, 8, 16, 32, 64.

  - File systems: Lustre and Expand.
    - Local storage device for Expand file system: NVMe as underlying storage (`/nvme`).
    - Block size for Expand file system: 128 MiB.

- File size: 32 GiB
  - Compute nodes: 1, 2, 4, 8, 16, 32, 64.
  - File systems: Lustre and Expand.
    - Local storage device for Expand file system: NVMe as underlying storage (`/nvme`).
    - Block size for Expand file system: 8 MiB.

3.3.1. TeraSort 512 GiB

In Figure 3.3.1, the preload (1) and flush (2) time for Expand can be seen, both in linear scale. Both plots follow the same trend as in the WordCount case. It can be seen how for 32 and 64 nodes the preload time experiences a slight scaling. This is because for this number of nodes, the file size per node is 16 GiB and 8 GiB, respectively, which may attenuate the effect of Lustre saturation.

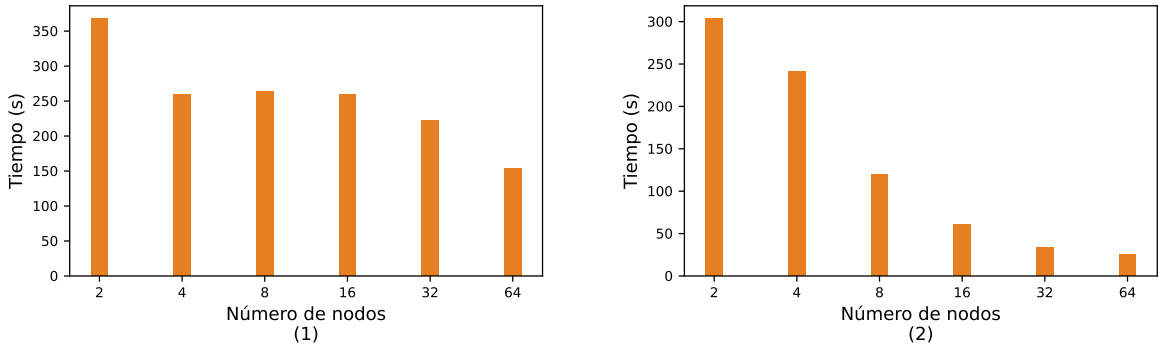

Figure 3.3.1: Expand preload (1) and flush (2) execution times in TeraSort (512 GiB).

In Figure 3.3.2, the measured times can be seen on a linear scale. In this case, the Expand performance is notably lower for a small number of nodes, however, as the number of nodes increases, the Expand times decrease notably and become lower than those provided by Lustre. On the other hand, again it can be seen how the saturation of Lustre affects the performance of Expand preload operations, becoming the main bottleneck of the evaluation.

Figure 3.3.3 shows the Speedup obtained during the evaluation. This Speedup has been obtained as follows:

$$S(N) = \frac{t(2)}{t(N)}$$

Where S(N) represents the Speedup for N nodes, N is the number of nodes and t(N) is the execution time measured for N nodes.

In the case of Lustre, it can be seen how the scaling is very limited. This is due to the fact that, added to the saturation produced by I/O operations, this benchmark has a high load of operations in memory, which prevents

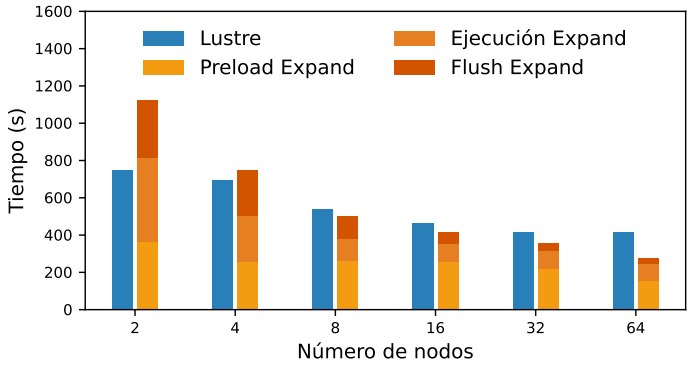

Figure 3.3.2: Lustre and Expand total execution times in TeraSort (512 GiB).

scaling with this file system. On the other hand, it can be seen how in the case of Expand, the impact of the memory load is mitigated thanks to the high performance offered by the I/O operations.

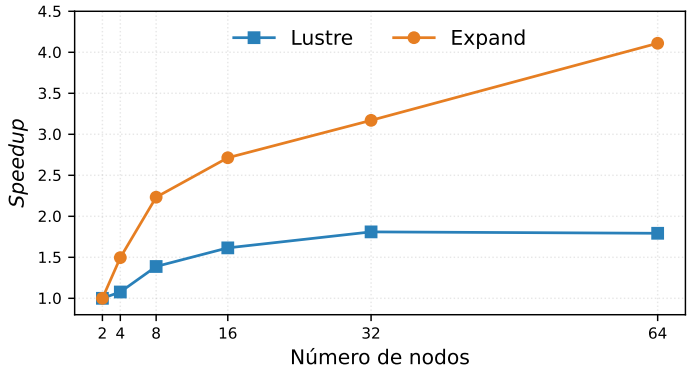

Figure 3.3.3: Lustre and Expand Speedup in TeraSort (512 GiB).

As was done with the evaluation of the WordCount, Figure 3.3.4 shows the Speedup graph without taking into account the Expand's preload time, since as seen previously, the main factor limiting its performance is the Lustre saturation during this operation. As can be seen, the growth is not as large as that observed for the WordCount tests due to the impact of in-memory operations. However, for 64 nodes, there is a 6-fold increase in the growth that Expand undergoes with respect to that of Lustre.

3.3.2. TeraSort 32 GiB

In Figure 3.3.5, the time of the preload (1) and flush (2) operations are shown on a linear scale. In contrast to the tests performed so far, the preload operation is not affected by Lustre saturation. This is because, despite the large number of requests made by the Lustre servers, given the low I/O load of each request, it does not saturate.

In Figure 3.3.6, the total times measured for the 32 GiB file are shown in linear scale. In this case it can be seen how the behavior of Lustre is quite similar to the tests performed with 512 GiB, however, the most significant impact can be seen in the performance of Expand, which by not being affected by the saturation of Lustre obtains a performance closer to that of Lustre with less number of nodes and the growth produced is

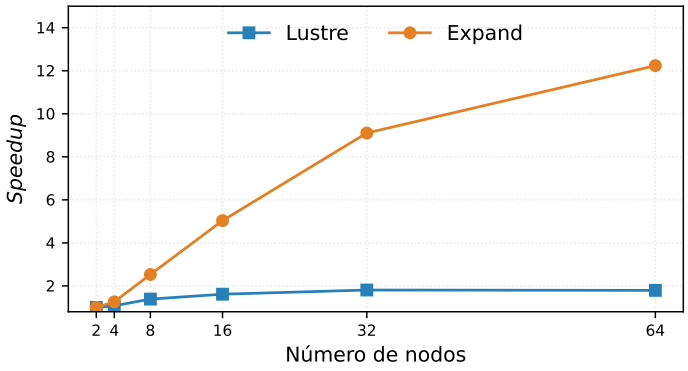

Figure 3.3.4: Lustre and Expand Speedup excluding preload time in TeraSort (512 GiB).

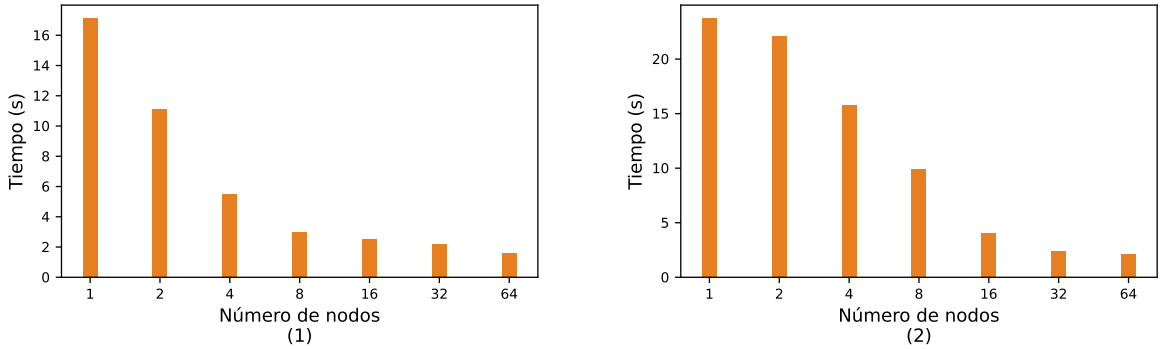

Figure 3.3.5: Expand preload (1) and flush (2) execution times in TeraSort (32 GiB).

much faster.

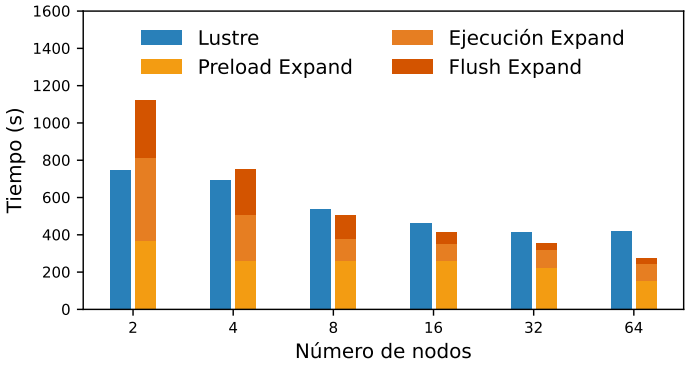

Figure 3.3.6: Lustre and Expand total execution times in TeraSort (32 GiB).

Figure 3.3.7 shows the Speedup obtained during the evaluation. This Speedup has been obtained as follows:

$$S(N) = \frac{t(1)}{t(N)}$$

Where S(N) represents the Speedup for N nodes, N is the number of nodes and t(N) is the execution time measured for N nodes.

In this evaluation, it can be seen that the Expand growth is still higher than that of Lustre. In this case, a deceleration can be seen in both systems, which is due to the smaller margin of improvement, since the volume of data to be processed is much smaller.

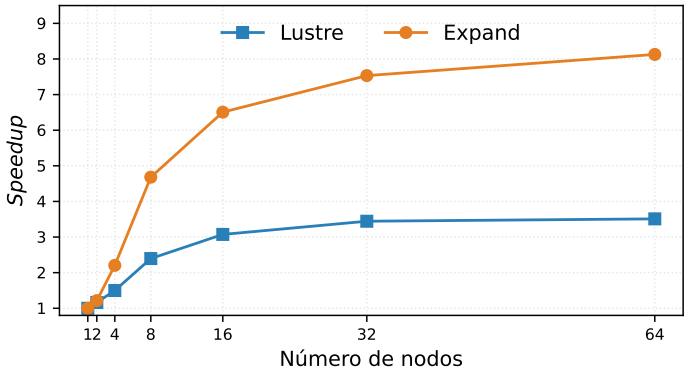

Figure 3.3.7: Lustre and Expand Speedup in TeraSort (32 GiB).

BIBLIOGRAPHY

[1] HPC4AI, *Especificaciones HPC4AI Laboratory*, https://hpc4ai.unito.it/documentation/. Accedido el 7-03-2025. [Online], 2025. [Online]. Available: https://hpc4ai.unito.it/documentation/ (visited on 03/07/2025).

[2] BSC, *MareNostrum 4 specification*, Accessed Feb. 22, 2025. [Online], 2025. [Online]. Available: https://www.bsc.es/marenostrum/marenostrum.

[3] CINECA, *Leonardo specification*, Accessed Feb. 22, 2025. [Online], 2025. [Online]. Available: https://leonardo-supercomputer.cineca.eu/hpc-system/.

[4] R. Sedona, G. Cavallaro, J. Jitsev, A. Strube, M. Riedel, and J. A. Benediktsson, "Remote sensing big data classification with high performance distributed deep learning," *Remote Sensing*, vol. 11, no. 24, 2019. DOI: 10.3390/rs11243056.