The role of storage target allocation in applications' I/O performance with BeeGFS

Francieli Boito, Guillaume Pallez, Luan Teylo

The role of storage target allocation in applications' I/O performance with BeeGFS

Francieli Boito, Guillaume Pallez, Luan Teylo*
Univ. Bordeaux, CNRS, Bordeaux INP, Inria, LaBRI, UMR 5800, F-33400 Talence, France
{francieli.zanon-boito, guillaume.pallez, luan.gouveia-lima}@inria.fr

***Abstract*—Parallel file systems are at the core of HPC I/O infrastructures. Those systems minimize the I/O time of applications by separating files into fixed-size chunks and distributing them across multiple storage targets. Therefore, the I/O performance experienced with a PFS is directly linked to the capacity to retrieve these chunks in parallel. In this work, we conduct an in-depth evaluation of the impact of the stripe count (the number of targets used for striping) on the write performance of BeeGFS, one of the most popular parallel file systems today. We consider different network configurations and show the fundamental role played by this parameter, in addition to the number of compute nodes, processes and storage targets.**

Through a rigorous experimental evaluation, we directly contradict conclusions from related work. Notably, we show that sharing I/O targets does not lead to performance degradation and that applications should use as many storage targets as possible. Our recommendations have the potential to significantly improve the overall write performance of BeeGFS deployments and also provide valuable information for future work on storage target allocation and stripe count tuning.

***Index Terms*—Parallel file system, parallel I/O, BeeGFS, I/O performance, performance evaluation, stripe count**

I. Introduction

In high-performance computing (HPC) platforms, the supercomputers, thousands of compute nodes work together to solve large problems. As these platforms evolved over the years, the I/O performance has grown at a slower pace than processing power, which caused many usually compute-intensive applications to start spending a large portion of their execution time on I/O operations. That problem is becoming even more relevant with the emergence of data-intensive HPC applications [4], [14], [18], [26].

At the core of a supercomputer's I/O infrastructure is the parallel file system (PFS). It is deployed over a set of dedicated servers shared by all the running jobs. These data servers, often called *Object Storage Servers* (OSS), store data in one or more *Object Storage Targets* (OST): logical volumes that represent storage devices. Files are typically separated into fixed-size portions and distributed across the storage targets in an operation known as striping. Then, parts of each file can be obtained from the targets in parallel for increased performance.

BeeGFS is such a parallel file system. According to data compiled by the OpenSFS group, in the *IO500 list from November 2019* [1], [2], BeeGFS was present in 20% of the submissions, second only to Lustre with 29%.

* The authors are presented in alphabetical order.

The performance observed when accessing a parallel file system is known to depend on the correct tuning of its parameters, including stripe size and count [8], [12], [19], [33]. **In this work, we are interested in the impact of the number of OSTs (the stripe count**). That is important because data is written to (or read from) the different targets in parallel, so more targets mean more parallelism, but too many targets may mean too much communication overhead. Differently from other PFS, such as Lustre, where users can easily configure the striping per file, in BeeGFS the stripe count is set by the administrator on a per-folder basis, hence a value must be chosen that is suitable for most applications. Moreover, since the PFS is shared, a large stripe count may force applications to share the storage targets, which could cause congestion. Our motivation behind studying this parameter is to quantify and qualify the congestion caused by sharing OSTs, and then to see how much congestion could be mitigated by some policy that adapts the stripe count of each application.

Some related work has focused on characterizing BeeGFS' performance in different contexts. Brzenski *et al.* [9] studied the impact of parameters such as the transfer size and the number of processes, seeking to optimize I/O performance of geophysical applications. Mills *et al.* [19] also evaluate BeeGFS as part of a solution to maximize the performance of scientific data transfer between different clusters. The authors compared different parallel file systems and concluded that BeeGFS showed the best performance in terms of throughput.

Nevertheless, to the best of our knowledge, Chowdhury *et al.* [12] were the only to evaluate the impact of the number of storage targets in this PFS' performance. From their results, obtained in an OLCF system called Catalyst, they concluded that increasing the stripe count has limited benefits for application performance. In their system, which has 24 targets in 12 servers, they point to 4 as the "reasonable" number of storage targets to use per application.

In this paper, **we provide a thorough study of the impact of the stripe count on write performance**. We aim (i) to reproduce some of their results on our local supercomputer; (ii) to give a more general and systematic methodology for conducting such evaluations on other systems, including in the presence of interference; (iii) to provide recommendations on tuning this value (i.e. to answer what should be the default stripe count in any BeeGFS system). **Our results contradict some of the recommendations by Chowdhury et al.** [12]: through an extensive analysis, we point some key parameters

that were neglected in their study and that can explain the difference in observations. Our main contributions are:

- Based on a comprehensive evaluation of BeeGFS in two versions of a system (using different networks), we characterize its write performance and especially the impact of the stripe count. Our findings contradict recommendations found in the literature and indicate how many storage targets should be used by default.
- We demonstrate the importance of network speed, often neglected when discussing I/O performance, and how it can completely change the observed behaviors. Our results also highlight variability and its importance when conducting an I/O performance evaluation.
- We present a methodology that can be applied in other systems to gather insights about their PFS. For instance, our conclusions led the system administrators of the machine we used, PlaFRIM, to change its default BeeGFS parameters. We estimate that change will transparently increase I/O performance of applications by up to 40%.
- By studying the impact of stripe count when multiple applications share the I/O infrastructure, we show that sharing storage targets to *not* negatively impact performance, and hence that policies that seek to adapt applications' stripe count would not improve write performance.

The remainder of this paper is organized as follows. Section II gives a brief overview of BeeGFS and its main components. Section III presents our evaluation methodology. Results and discussions are presented in Section IV and Section V talks about related work. Finally, Section VI concludes the paper and introduces future directions.

II. BeeGFS

BeeGFS is an open-source parallel file system originally developed at the Fraunhofer Center for High-Performance Computing. The system was designed with a strong focus on performance, scalability, and usability [5], [9]. It can be deployed at any Linux-based computing system and works with several local file systems, such as ext4, xfs, or zfs. As depicted in Figure 1, the components of BeeGFS are divided into four categories: client (blue, on the top), management (green, on the left), metadata (yellow, on the bottom-left), and storage (red, on the bottom-right).

The Management Server (MS) maintains a list of all system components, including their status, capacity, and localization. It is responsible for ensuring that the PFS parts can find each other. As shown in the bottom left of Figure 1, the metadata management in BeeGFS uses two distinct components. The first one, called *Metadata Server* (MDS), is a service responsible for handling the metadata operations over an exclusive portion of the file system tree. An unlimited number of MDS can be used, operating on different portions. The second component is the *MetaData storage Target* (MDT), which stores the metadata on a storage device, typically an SSD. Each MDS can have precisely one MDT.

Similarly, data storage is also composed of two components: the *Object Storage Servers* (OSS) and the *Object Storage Targets* (OST). The OSS is the service responsible for keeping files' data, while the OST handles the actual storage to devices through a local file system. The storage device used by an OST is typically a RAID-6 array of six to twelve hard disks [**?**]. As can be seen in Figure 1, differently from the MDS, each OSS can have more than one OST.

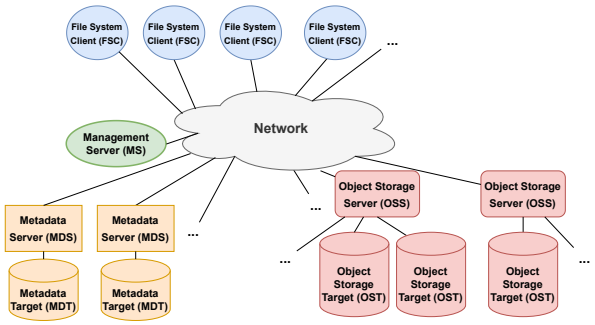


Fig. 1: Logical components of the BeeGFS software architecture (figure inspired by [12, Figure 1]).

As stated in the previous section, when a new file is created, it is *striped* and distributed across the storage targets. In BeeGFS, striping is defined by two parameters set on a per-directory basis: the *stripe count*, which defines the number of storage targets to use; and the *stripe size* (the size of each portion of the file). By default, the OSTs used to store each file are randomly chosen. However, other heuristics can be used. For example, the targets can be selected in a deterministic round-robin fashion. As we will see in Section IV, depending on the *stripe count*, the used target selection heuristic can contribute to the observed I/O bandwidth.

Finally, the *File System Client* (FCS) is a kernel module that allows for mounting the remote file system and that exposes some useful functions. Note that the term "server" in BeeGFS does not refer to a physical machine but to a Linux process. That differentiation is important because different servers are generally executed on the same physical machine (for example, an OSS and a MDS on the same physical server). Still, throughout this paper, we use the term "storage server" to refer to a physical machine running an OSS. In the same way, the term "computing node" refer to a machine where one or more processes belonging to an application are running.

III. Methodology

In this section we describe the experimental methodology we applied for this study of BeeGFS performance: the platform (Section III-A), the benchmarking tool and the used parameters (Section III-B), and the execution protocol (Section III-C).

All scripts used for our tests as well as their results are available and documented at https://gitlab.inria.fr/hpc_io/beegfs_evaluation.

A. Experimental environment

All experiments presented in this work were conducted in PlaFRIM, a 192-nodes experimental platform located at the Inria Bordeaux research center. Specifically, we used the Bora cluster, whose nodes are each powered by two 18-core Intel

Xeon processors, 192 GiB of RAM memory, and run CentOS 7.6.1810 with Linux kernel v3.10.0-957.el7.x86_64.

In addition to users' homes (on NFS), a parallel file system storage with BeeGFS [5] v.7.2.3 is available for all of PlaFRIM's clusters. BeeGFS is deployed over two hosts, used for both data and metadata. Each host executes one OSS with four OSTs and one MDS. Each OST uses 12 Toshiba AL15SEB18EQY HDDs, each with 1.8 TB of capacity and running at 10,000 RPM, the 12 organized in RAID-6. On the other hand, each MDS has one MDT with two Samsung MZILT1T6HAJQ0D3 SSDs of 1.6 TB organized in RAID-1. The total data storage capacity of the deployed system available to the clients is 131 TB. In PlaFRIM 's current setup, files are written with *stripe count* of 4 and *stripe size* of 512 KiB, and the OSTs are selected in a round-robin fashion. This OST allocation heuristic is *not* the BeeGFS' default: the vendor's team set it up when delivering the system (after benchmarking it). The impact of this choice in our results and what they would be with other heuristics will be discussed in Section IV.

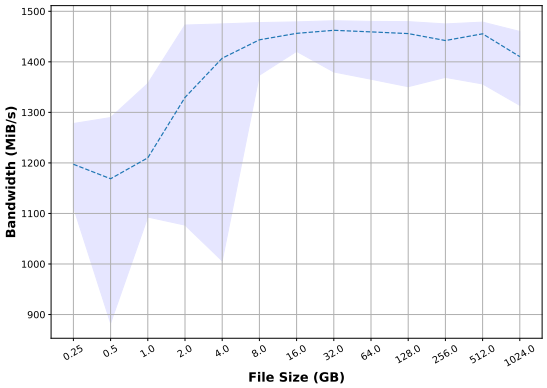
PlaFRIM's nodes and storage servers are connected by a 10 GBit/s Ethernet network. Additionally, Bora nodes also share a 100 GBit/s Omnipath network that includes the PFS. Since PFS performance is limited by the slowest component in the I/O path, these networks represent two distinct scenarios of execution. In *Scenario 1*, using Ethernet, the network speed is slower than the storage components, limiting I/O performance. In *Scenario 2*, the speed of the Omnipath is greater than that of the storage components, so the latter are the most important factor for performance. In both scenarios, the Bora nodes were connected directly to the BeeGFS hosts through a switch (models Dell S4148F-ON and Dell H1048-OPF for Ethernet and Omnipath, respectively).

B. Benchmarking tool

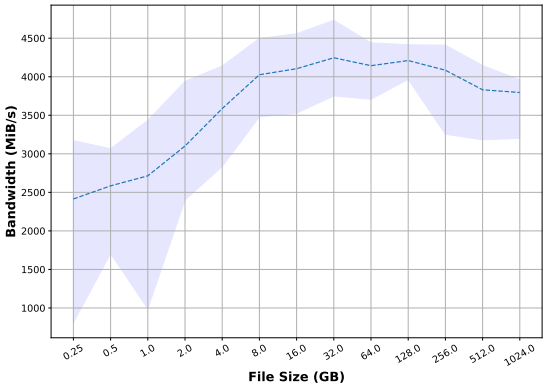
We generated all tests using IOR [7] version 3.4 compiled with GCC v.4.8.5. IOR is a benchmark tool that measures the performance of I/O operations considering different parameters such as the file size, the transfer size and the number of segments. On the survey presented by Boito *et al.* [8], IOR was the most used benchmark for research on HPC I/O. It is also the main benchmark used for the IO500 list [1].

Each experiment consists of multiple IOR executions, i.e. we do not use the "-i" option that asks IOR to repeat the experiment multiple times. We chose to do that to avoid warm-up effects. Moreover, we used the POSIX interface and 1 MiB transfer size, which is aligned to stripe size and large enough (compared to the default stripe size) to require more than one OST to be accessed for each request.

We focus on write performance because that is when the stripe count is more relevant: once files are written, changing the stripe count requires data migration between targets. Although extending our conclusions to read performance will be the subject of future work, based on the results by Chowdhury *et al.* [12], we expect the observed behaviors to be the same.



(a) *Scenario 1*: network is slower than storage



(b) *Scenario 2*: storage is slower than network

Fig. 2: Impact of the data size on I/O bandwidth. A stripe count of 4 was used in this experiment. The blue dotted line is the mean of 100 repetitions, and the shadow shows the difference between the maximum and minimum values. The y-axes do *not* start at zero and are different in each plot.

To limit the impact of metadata overhead in our results — because we are only interested in the number of OSTs — we used a shared-file strategy (N-1). In each test, application processes write to contiguous portions within a shared file. Contiguous access was selected because we wanted to analyze our system in a peak performance state, in order to isolate the impact of the studied parameters on performance.

1) Amount of data: To reach the aforementioned peak performance state, it is important to select a "large-enough" data size (which corresponds to the file size when using the N-1 strategy). That is the case because small accesses are more impacted by latency than by data access bandwidth.

To find that "large-enough" size, we conducted a first experiment using 32 processes on 4 nodes. Figure 2 presents the obtained results. The first thing to notice is that small data sizes have lower overall performance but also higher variability, which is another reason why it was important to properly select the data size. We can see that in both scenarios

performance stabilizes starting from a size between 16 and 32 GiB. Based on these results, we used a total size of 32 GiB for all other experiments.

C. Execution protocol

Since PlaFRIM is a production system, we designed an execution protocol aiming to minimize the influence of (i) I/O operations issued by other users during our tests; (ii) transient events in the machine that would temporarily lower network and/or I/O performance; (iii) caching on the clients and on the file system servers; and of (iv) the well-documented high variability of I/O performance [11], [17], [21], [24], [27].

In order to do that, we try to cover multiple system states for each of the tested configurations by following these steps:

1) we generated a list of all benchmark runs containing 100 repetitions of each of the different experiments;
2) that list was then divided into blocks of ten executions;
3) the list of blocks was executed (one test run at a time so we do not influence our own results) in a random order;
4) between blocks, a randomly selected waiting time (between 1 and 30 minutes) is imposed.

IV. Results

This section presents our analysis of BeeGFS performance. We evaluate the impact on performance of the number of compute nodes in Section IV-A and of processes per node in Section IV-B. This initial study has the goal of determining the right parameters to be used when evaluating the impact of the number and placement of storage targets, but it is presented here mostly because it leads to interesting conclusions and justifies differences between our conclusions and the ones by Chowdhury *et al.* We then discuss the impact of number and placement of OSTs in Section IV-C, and of concurrent accesses (by multiple applications) in Section IV-D.

A. Computing nodes

We expect I/O performance to increase (up until some point) with the number of used compute nodes and/or processes because (i) a storage device's peak performance is usually only reached at a certain parallelism level (number of concurrent accesses), and (ii) not using enough nodes may limit network performance [28]. Figure 3 illustrates this second argument. Assuming all links have the same capacity B, when N nodes are used to access M servers from the PFS, *network* performance is limited by $M \times B$ only when $N \geq M$, otherwise the limitation is $N \times B$.

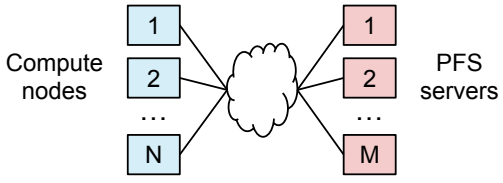
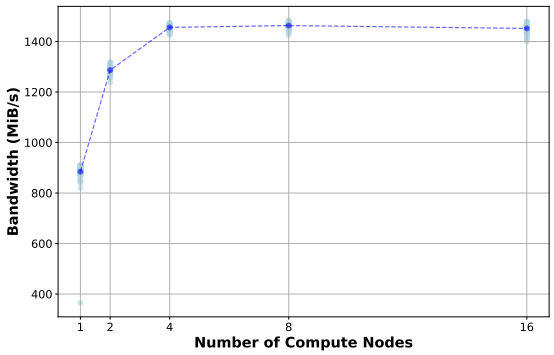
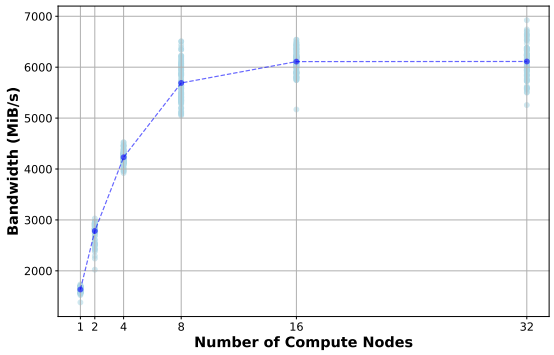


Fig. 3: N compute nodes concurrently access M OSSs



(a) Scenario 1: network is slower than storage



(b) Scenario 2: storage is slower than network

Fig. 4: Evolution of the I/O bandwidth according to the number of compute nodes in scenarios 1 and 2 using 8 processes per node. The dotted line connects the mean bandwidth values, while the blue dots represent individual executions. The figures are not in the same scale of bandwidth, the y-axes do *not* start at zero, and **the x-axes are different in the two plots**.

For these reasons, we first evaluate how the number of computing nodes impacts the I/O bandwidth in both scenarios. For this experiment, we used eight processes per node and varied the number of nodes. The total file size is always of 32 GiB, therefore the amount of data written per process is adapted accordingly. For example, with one node each of the eight processes write 4 GiB, and with eight nodes the 64 processes write 512 MiB each. For all experiments in these first part (Sections IV-A and IV-B), the default stripe count of 4 was used.

As can be seen in Figures 4a and 4b, in both scenarios the bandwidth initially increases when more nodes are used to perform I/O. In scenario 1, the bandwidth goes from an average of $\sim$880 MiB/s with one node, and reaches a *plateau* of $\sim$1460 MiB/s when $N = 4$. In the second scenario, it goes from $\sim$1631.5 MiB/s with one node and reaches a *plateau* of $\sim$6100.2 MiB/s with 16 nodes. One thing to notice is that the bandwidth observed in the second scenario is generally higher than for the first scenario because of the faster network. The second scenario shows the speed of the storage infrastructure of PlaFRIM. For the same reason (no longer being limited by the network), in the second scenario more nodes are required

to achieve the peak performance.

If we ignore all overhead possibly caused by other elements in the I/O path, we know that the I/O bandwidth in scenarios 1 and 2 cannot exceed the aggregated bandwidth of the two links leading to the two storage servers ($\sim$2500 MiB/s and $\sim$25000 MiB/s for scenarios 1 and 2, respectively). However, as shown in Figure 4, the I/O bandwidth is always far below those limits in both cases. In scenario 1, the main reasons for such low bandwidth are related to the OST allocation. We will discuss that in Section IV-C, but for now, let us concentrate on the behavior of the bandwidth regarding the number of computing nodes.

Lesson learned #1

Our results show that the number of compute nodes can limit I/O performance regardless of the network speed. Indeed, in scenario 2, where the storage performance is what defines performance, more compute nodes were required than in scenario 1, and the impact of using them was heavier (270% instead of 64%). Still, the literature shows that many scientific applications use a single node for I/O [26].

Therefore, the number of compute nodes needs to be considered in the performance evaluation of other parts of the I/O path. If not enough nodes are used, the low performance may hide some interesting behaviors. That is probably the case of one of the results presented by Chowdhury *et al.* [12, Figures 5 and 6], where the authors concluded the impact of the stripe count on the bandwidth was negligible while evaluating with a single compute node. We believe that the insufficient number of nodes caused the actual effect of the number of OSTs on their experimental platform to not appear in their results.

Lesson learned #2

Finding the number of computing nodes that leads to the maximum I/O performance should be the first step in evaluating a PFS. Otherwise, the low bandwidth can hide the effects caused by other parameters, such as the number of OSTs.

B. Number of processes per node

As we use two storage servers, we expect two computing nodes to be enough to reach peak performance, at least when limited by network performance (scenario 1). However, as shown in Figure 4, four nodes are used to achieve the plateau in the first scenario, and 16 in scenario 2. In these experiments, since the same amount of data is always used, the only difference between different numbers of nodes is that more requests are generated concurrently to the servers, i.e. there is more available parallelism. That means that the increased performance observed when using more than two compute nodes comes from the higher parallelism at the storage system.

A natural conclusion from that observation would be that, as long as we use more than two nodes, we could increase the number of processes per node to decrease the number of nodes, since that would generate a similar parallelism level. To test that hypothesis, we included experiments using 16 processes per node. Results are presented in Figure 5, and show that the behavior regarding the number of compute nodes does *not* change. In fact, the bandwidth remains very similar, with a slight degradation in scenario 2.

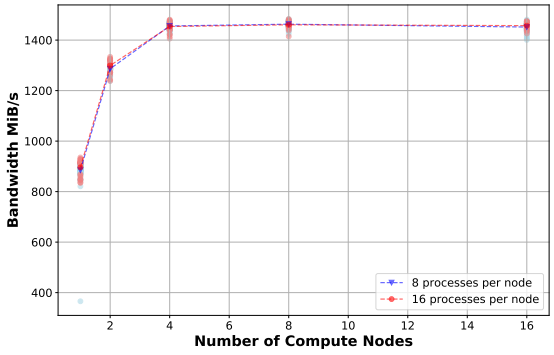
One possible explanation of this is intra-node contention [13], caused by the processes inside each node competing for access to the network interface, memory, BeeGFS client, etc. More investigation is needed to determine if that behavior comes from limitations in the BeeGFS client's ability to handle parallel accesses. Chowdhury *et al.* [12, Figures 5 and 6], reported a similar behavior regarding the number of processes (although they evaluated it with a single compute node).

Lesson learned #3

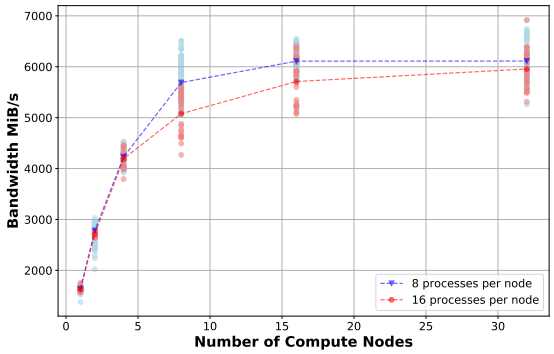
The numbers of processes and nodes have independent effects on performance, meaning that both must be considered when evaluating I/O performance.

C. Object Storage Targets

Next, we evaluate the impact of the stripe count on performance. Based on the results observed in the previous sections,

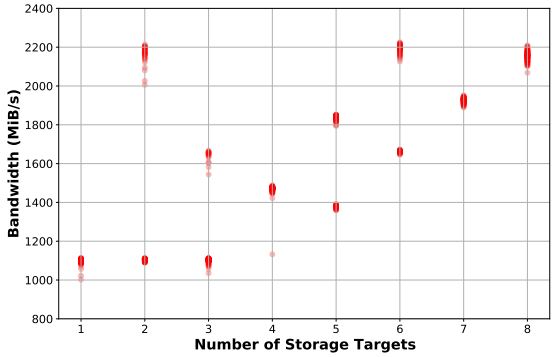


(a) Scenario 1: network is slower than storage

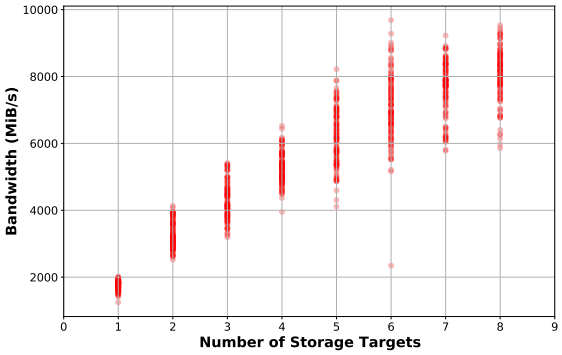


(b) Scenario 2: storage is slower than network

Fig. 5: Impact of the number of compute nodes on I/O bandwidth with different numbers of processes per node. The dots represent individual executions, while the dotted lines connect the mean values. The x-axes and y-axes are different in the two plots, and the y-axes do *not* start at zero.



(a) Scenario 1: network is slower than storage (8 nodes)



(b) Scenario 2: storage is slower than network (32 nodes)

Fig. 6: I/O bandwidth for different numbers of storage targets (stripe count). The red dots represent the bandwidth values of the individual 100 executions. The figures are not in the same scale of bandwidth and the y-axes do *not* start at zero.

we use 8 compute nodes for scenario 1 experiments, and 32 nodes for scenario 2, in both cases using 8 processes per node.

Figures 6a and 6b present the I/O bandwidth observed with all possible numbers of targets in PlaFRIM. We can see this parameter can have a heavy impact, changing performance from ∼1100 MiB/s to ∼2200 MiB/s in the first scenario, and from ∼1760 MiB/s to ∼9000 MiB/s in the second one. However, we can also notice that both cases present very distinct behaviors.

1) Scenario 1 (performance is limited by network): In the first scenario, two distinct effects stand out. The first one is that in most cases, the reported I/O bandwidth presents a bi-modal behavior. That is the case when 2, 3, 5, and 6 OSTs are used. The second behavior to notice is that the peak performance — ∼2200 MiB/s — is reached only when the stripe count is equal to 2, 6, or 8. Therefore, in scenario 1, the default striping pattern with 4 OSTs (also recommended by Chowdhury *et al.* [12]) keeps the I/O performance of PlaFRIM below 50% of the peak.

To understand this behavior, we analyze the relationship between target selection and I/O bandwidth. As discussed in Section III-A, PlaFRIM's storage infrastructure has two servers (OSS), each with four targets (OSTs). Let S_i be the

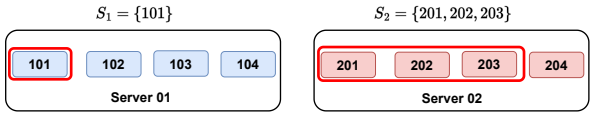


Fig. 7: Example of an OST allocation with four targets and placement of $(1, 3)$

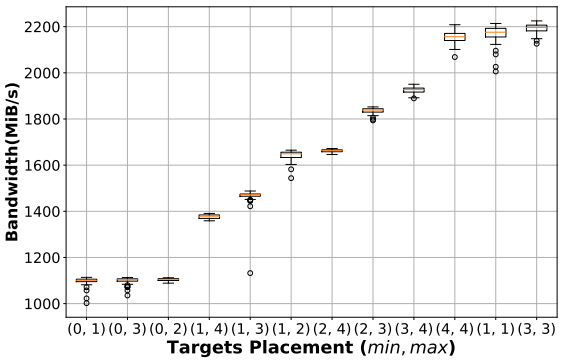


Fig. 8: Box-plots of performance according to OST allocation in scenario 1 (network is slower than storage). Generated from the data presented in Figure 6a. The y-axis does not start at zero.

set of OSTs used in the ith server during an I/O operation. We represent the OST allocation by the number of targets selected in each server as (min, max), such that $min = minimum(|S_1|, |S_2|)$ and $max = maximum(|S_1|, |S_2|)$. For example, Figure 7 illustrates a possible allocation of four targets: one in the first server and three in the second one, which we represent as $(1, 3)$.

Using this (min, max) notation, we separated data from Figure 6a by their OST allocation and generated the box-plots presented in Figure 8. Interestingly, we can see that performance increases with the min/max ratio. Moreover, the actual number of targets does not have any impact, seeing as $(0, 1)$, $(0, 2)$, and $(0, 3)$ have very similar performance. The same can be said for $(1, 2)$ and $(2, 4)$, and to $(1, 1)$, $(3, 3)$, and $(4, 4)$. Essentially, the highest performance is reached when the number of targets is the same in both servers. In contrast, the lowest performance happens when just one of the servers is used.

To illustrate that behavior, we consider the case where two targets are used. In this case, there are two possible allocations: $(0, 2)$ and $(1, 1)$. Figure 9 shows what happens in both cases: since in scenario 1 the performance available from each server is limited by the network link leading to it, using both servers in a balanced way (i.e. the same amount of data is written to both) leads to the best possible performance.

The bi-modal behavior observed for some numbers of targets come from different allocations that are made by BeeGFS in different repetitions of the experiment. Although the default striping with 4 OSTs could have the balanced placement $(2, 2)$, we do not see it in Figure 8 because it never happened in 100 repetitions of our experiment. The round-robin heuristic used in PlaFRIM always makes a $(1, 3)$ allocation: (101, 201, 202, 203) or (204, 102, 103, 104). Comparing the performance

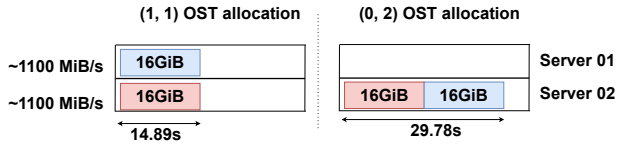


Fig. 9: Performance for Scenario 1 (network is slower than storage) when writing 32 GiB to two storage targets depending on the allocation. The x-axis represents time, and the y-axis represents the bandwidth available from each server (i.e. the capacity of the network link to that server).

of that allocation to the $(3, 3)$ that always happens with six targets, the latter increases bandwidth by more than 49%.

Some of the observed behaviors happen because of the use of the round-robin OST allocation heuristic. For example, as previously stated, a stripe count of 4 always results in a $(1, 3)$ allocation. If a random selection of OSTs were to be used instead, then all other allocations would be possible, including the balanced $(2, 2)$ that would result in peak performance. Nonetheless, it is important to notice that does *not* mean that the stripe count of 4 would always reach peak performance. Its performance would actually have a high variability, with the best case being as likely as the worst case.

Lesson learned #4

When the network speed limits the I/O performance, the main factor that impacts the I/O bandwidth is not the number of used targets, but rather the load balance of target allocation among the storage servers. A selection heuristic that picks the same number of targets in the storage servers would be the best choice. Without changing the heuristic, the maximum number of storage targets (eight in our case) can be selected as the default stripe count in order to achieve peak performance every time.

There is another important thing to notice from results in Figure 6a : the impact of our experimental methodology in our conclusions. All our experiments were repeated 100 times and we looked at all the points, not only their mean or median. If we had executed these tests only once or even a few times, because some (min, max) allocations are more frequent than others, or even if we had simply calculated and plotted the mean bandwidth without looking at the data, we could tell a different (and inaccurate) story.

Lesson learned #5

It is always good practice to repeat experiments multiple times, but that is especially true when evaluating I/O because of the large number of variables that affect results. Moreover, one should be careful when summarizing data points by their mean as that can hide interesting and relevant behaviors.

2) Scenario 2 (performance is limited by the storage system): As we can see in Figure 6b the bi-modal behavior seen in scenario 1 is not present in scenario 2. Instead, the

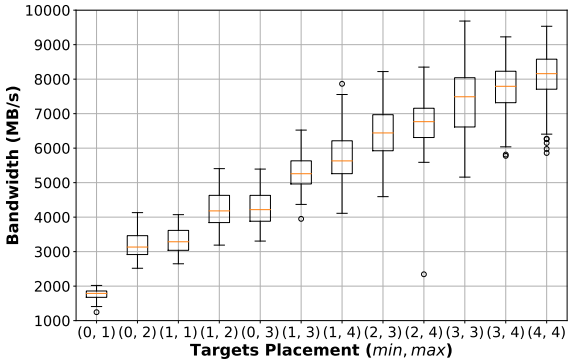


Fig. 10: Box-plots of performance according to OST allocation in scenario 2 (storage is slower than network). Generated from data presented in Figure 6b. The y-axis does *not* start at zero.

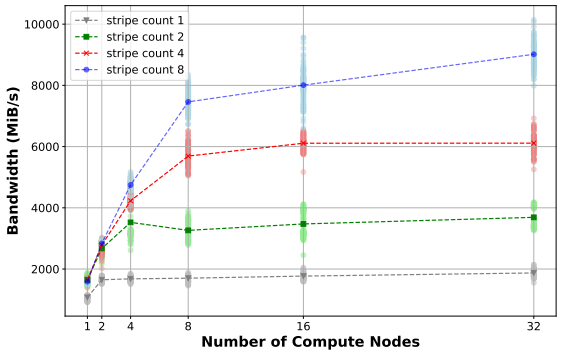


Fig. 11: Mean values of the I/O bandwidth in scenario 2 according to the number of compute nodes, and considering multiple stripe counts.

bandwidth increases almost linearly with the number of OSTs and presents a high variability. When increasing from 1 to 8 storage targets, the mean bandwidth was increased by more than 350% (from $\sim$1764 MiB/s to $\sim$8064 MiB/s), while the standard deviation increased in more than 460% (from 139.8 to 787.9).

That variability is caused by the performance variation of the storage devices [10]. Once the performance of the I/O is limited by the storage devices' speed and not by the network, any performance variation on those devices will reflect in the I/O bandwidth.

The OST allocation for scenario 2, shown in Figure 10, confirms that the number of OSTs is the most important parameter in this case. Figure 10 also shows that well-balanced placements, such as $(1, 1)$, $(3, 3)$, and $(4, 4)$, have in general a better performance than unbalanced placements with the same number of targets. For example, the average bandwidth for the well-balanced placement $(3, 3)$ is 10.15% higher than the unbalanced placement $(2, 4)$. Thus, although the impact of the placement is not so evident as in scenario 1, selecting the same number of OSTs in each storage server is still the better choice.

In this section, we used 32 compute nodes for scenario 2

while Section IV-A had shown 16 to be enough. That choice was made because of the results presented in Figure 11, where we repeated those experiments in scenario 2 but with different numbers of OSTs (4 had been originally used). This shows that with more storage targets higher peak performance is available, but that performance can only be achieved with more compute nodes.

Lesson learned #6

In contradiction with earlier findings [12], adding more OSTs in BeeGFS does indeed lead to more performance in the case where I/O performance is *not* limited by the network. Moreover, the number of computing nodes required to reach peak performance depends on the stripe count. Although at a small impact, when compared with scenario 1, the well-balanced target allocation is also the better choice in scenario 2. As in scenario 1, the maximum number of targets (8 in our case) allows for the best performance that does not depend on target allocation.

D. Impact of concurrent applications sharing OSTs

The evaluation of the impact of stripe count on performance for both considered scenarios has concluded that performance is linked to the number of OSTs (for scenario 2) and their allocation in the servers (especially for scenario 1 but also for 2). That being the case, a good strategy could be to always use the maximum number of available targets, since that (i) allows applications running on their own to achieve peak performance; (ii) eliminates the impact of OST allocation across servers; and (iii) is a simple recommendation that can be followed without requiring similar analyses to be conducted on other machines. Nonetheless, that strategy has a disadvantage: by allowing all applications to use all storage targets, we maximize how many targets they share. Therefore, in this section, we investigate the impact of that on performance. We focus on Scenario 2, where performance is limited by the storage infrastructure, because we are interested in the impact of sharing the OSTs.

Figure 12a presents results obtained when running two concurrent applications, each using eight different compute nodes (they do *not* share nodes), and increasing the number of OSTs each application uses. Equation 1 explains how we calculated aggregate bandwidth for these experiments, where A is the set of concurrent applications, $start_i$ and end_i are the start and end times of application $i \in A$ and vol_i is the amount of data written by that application.

$$\frac{\sum_i vol_i}{\max\limits_{\forall i \in A}(end_i) - \min\limits_{\forall i \in A}(start_i)} \quad (1)$$

When the stripe count is 2, applications never, in 100 repetitions, shared the same targets. In that situation, as can be seen in Figure 12a, the aggregate bandwidth is compatible with what was previously observed for a single application using 16 nodes and 4 targets. With 4 and 8 OSTs being used per application, the aggregate bandwidth continues to be very similar — and even slightly higher — than what was achieved by a single application with twice the number of nodes and targets. In other words, even when all the targets were shared by the two applications, global performance was not degraded. The same behavior can be observed in Figures 12b and 12c, where results obtained with 3 and 4 concurrent applications are presented.

Another thing to notice from Figure 12 is that performance achieved by each application is lower than what was observed for the same application running by itself, and that difference increases with the stripe count. We conclude that slow-down in individual performance comes from sharing the available bandwidth, and *not* from contention at the shared targets, because that also happened in the tests where each application used two targets (slow-down of up to 20%), and in those tests they never shared OSTs. The fact that the difference is higher for more applications and higher stripe counts simply comes from the fact that more applications are sharing the available bandwidth, and that with a higher stripe count the single-application baseline is faster.

With two applications using four storage targets each, we can separate results in two different cases. Because PlaFRIM's BeeGFS has only two possible OST allocations for a stripe count of 4 — both $(1, 3)$ — the two applications either did not share targets (which happened in approximately two thirds of our test repetitions), or they shared all four targets (the remaining one third of results). Figure 13 shows individual performance observed in both cases. A Welch two-sample t-test was applied to compare the two groups (after testing normality with the Kolmogorov-Smirnov test and assuming different variances) and resulted in a p-value of 0.9031, which does not allow to reject the hypothesis of the two means being the same (in other words, *we cannot conclude they are significantly different*).

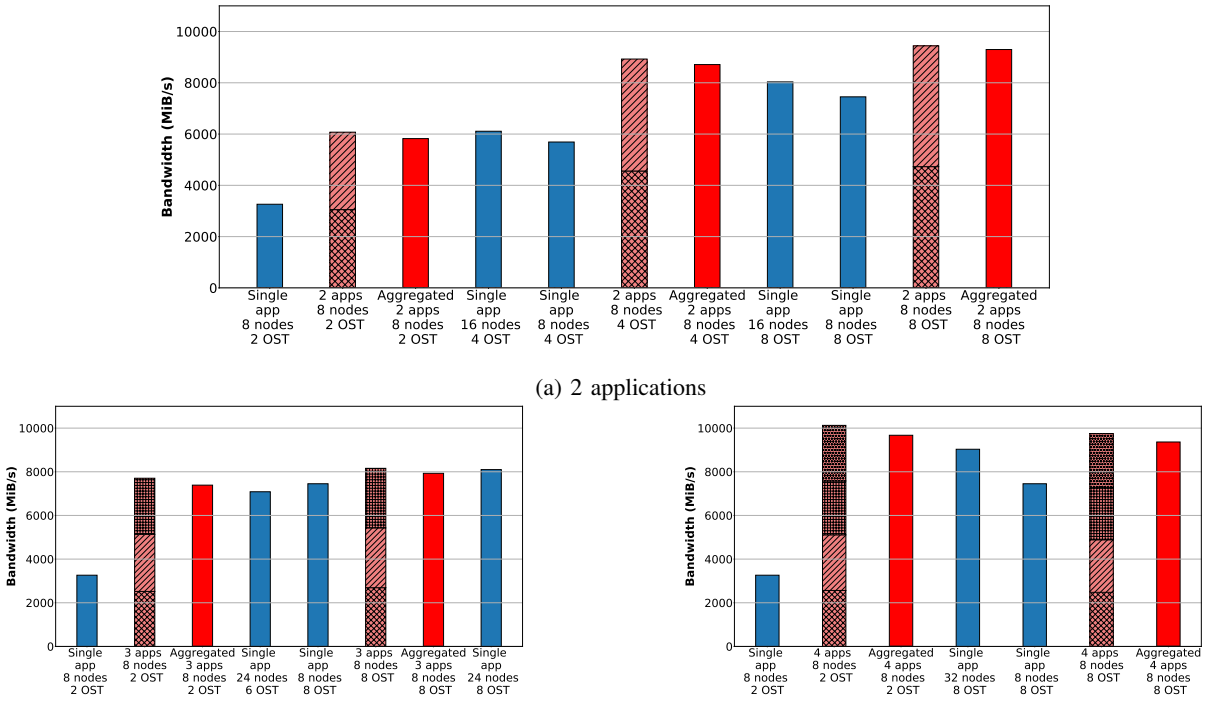
Lesson learned #7

Our results suggest that sharing OSTs among concurrent applications does not significantly impact I/O performance. Of course, we expect performance degradation with a large number of concurrent applications, but that situation — having many concurrent applications that write large amounts of data at the same time — is not very common in a supercomputer [28].

It is important to notice we are *not* saying I/O performance in general is never harmed when multiple applications compete for the PFS, but that this degradation does not come from sharing OSTs. Indeed, I/O interference has been shown to be connected to metadata intensity [31], network behaviors [32], and sharing other parts of the I/O stack [21].

V. Related Work

I/O performance when accessing a parallel file system depends on the non-trivial interplay of large number of parameters [27]. For this reason, some papers try to use machine learning techniques to predict it [23], [29].



(a) 2 applications

(b) 3 concurrent applications

(c) 4 concurrent applications

Fig. 12: Performance of concurrent applications (in red), with different numbers of OSTs per application, compared to single application executions with similar parameters (in blue). The stacked bars show the individual bandwidth of the applications, and their aggregate bandwidth was calculated with Equation 1. Individual performance should be compared to the left, and total/aggregated to the right. Each bar is the average of 100 executions, all results can be seen in the git repository.

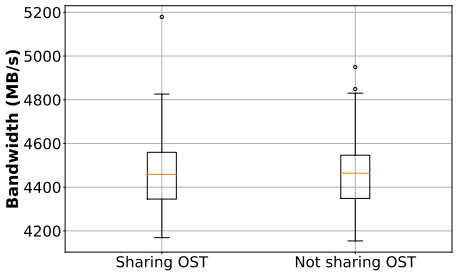


Fig. 13: Individual application performance when two concurrent applications access 4 OSTs each, separated by cases where they are all the same or all different.

A considerable amount of work has studied the performance of the Lustre parallel file system [3]. Shan and Shalf [22] focused mainly on application-related parameters, such as the transfer size and the number of processes. Wang et al. [27] used logs to study the Lustre deployment of Cori, aiming to identify applications' I/O bottlenecks. They concluded the aspects with most impact on performance are: number of compute nodes, processes, and OSTs; and data size. They also pointed unbalanced workload on OSTs as one of the root causes for poor job performance. For the same system, Kim *et al.* [15] proposed an algorithm that uses information from previous application runs, such as number of processes and of I/O requests, to dynamically adjust Lustre striping for each job. In their results, performance was improved by 50%.

In [6], Behzad *et al.* present an auto-tuning framework to adapt I/O stack parameters including Lustre stripe size and count. For that, it keeps a database of I/O patterns, extracted from applications, and non-linear regression models to find the best parameters. Differently from their work, which is more application-centered, our objective was to search for the best general-purpose default stripe count, without relying on previous knowledge about the applications, I/O profiles that can be inaccurate, regression models that have some error, etc.

Like us, Lawrence *et al.* [16] found that the best stripe count for a single shared file is to use as many OSTs as possible. Wan et al. [24] measured write performance when accessing a single OST by an increasing number of concurrent applications, and observed there was no significant degradation. They also pointed that load balance among OSTs is important. For a result where bandwidth was multi-modal, authors argued it was due to interference from other jobs. Wang et al. [25] profiled two large-scale Lustre deployments and observed that 96% of jobs use the default stripe count, which is small in these systems. In addition to studying a different file system, although we reach some of the same conclusions, our work differs from these because we further study the impact of OST allocation, the role of network performance, and explain high variability and multimodal behaviors. Moreover, in BeeGFS

the users cannot change stripe configuration as easily, which makes the correct choice of the default values even more important.

Xu et al. [30] proposed a tool for monitoring performance of the Lustre file system, and then used it to evaluate performance of MPI-IO collective read and write operations. They have improved read performance by making each aggregator access less storage targets, as they observed the concurrent accesses led to more disk seeks. This highlights the importance of the application characteristics for performance. In other words, even with well-tuned PFS parameters performance can still be jeopardized by an inadequate access pattern.

Being more recent, BeeGFS have been evaluated in a few studies in the related literature. Morganti et al. [20] studied — with benchmarks and bioinformatics applications — the performance of BeeGFS deployed on low-power SoCs. They did not focus on PFS parameters and kept the default values for their experiments. Mills *et al.* [19] evaluated the components involved in long-distance transfers of large datasets between clusters, including BeeGFS. They experimented with different numbers of OSSs (concluding that the higher, the better), but used a single compute node (which may cause behaviors to be hidden, as we discussed in Section IV-A) and did not study the impact of other parameters such as the number of storage targets, of compute nodes, and the network speed.

Brzenski *et al.* [9] aimed at optimizing I/O performance of geophysical applications using PnetCDF to access BeeGFS. Therefore, they evaluated different parameters, including combinations of MPI-IO hints. They were focused on a specific type of application and not on a general case, like we are. Still, they presented a result with three different stripe counts where the higher was the better. Nonetheless, differently from us, they did not consider target allocation, the number of compute nodes and of processes per node, or network speed.

As previously said, to the best of our knowledge, Chowdhury *et al.* [12] is the closest work to ours, since they evaluated the impact of the number of storage targets on the BeeGFS performance. However, as we showed in this paper, we did not reach the same conclusions. Our rigorous methodology, and the fact we studied different network speeds and target allocation, allowed us to identify important behaviors. Mainly, we showed that the maximum possible stripe count should be use for peak performance, and then showed that sharing targets do not significantly degrades application I/O performance.

VI. Conclusion

In this work, we investigated the impact of storage target allocation (number and placement) on write performance with BeeGFS. Our main goal was to extend the state of the art [12] (i) by obtaining guidelines that would be generic for any system, and (ii) by investigating the possible role of a target allocation policy. To do that, we conducted a comprehensive performance evaluation using the well-known IOR benchmark tool and studying the impact of the data size, the number of compute nodes and of processes per node, the network speed, the stripe count and the placement of OSTs in OSS. All of these parameters were shown to play important roles.

Notably, we showed that using the maximum possible stripe count is the best strategy, as lower counts are affected by the placement of OSTs among OSSs. Moreover, in the scenario where performance is limited by the storage components, performance increases with the number of OSTs. That conclusion contradicts previous observations that advised against that by citing limited performance improvements by adding OSTs (which were observed in conditions that would hide their impact) and a need to avoid sharing of targets by concurrent applications. Nevertheless, we did not observe performance degradation with up to 4 I/O-intensive concurrent applications.

In this analysis, we used a total data size large enough to reach the system's peak performance. The literature says that lower stripe counts could be better to small file sizes [25]. Still, we believe the default value should benefit large accesses, which are the most bandwidth-critical ones.

We estimate that the change in the default stripe size of the system we used transparently increased applications' I/O performance by more than 40%. In addition to exploring parameters often neglected in the literature, such as the network speed, our results also highlight the importance of a rigorous experimental methodology. The lessons learned from our study provide important guidelines not only for BeeGFS configuration, but for PFS performance evaluation in general.

Our conclusions are relevant for small to medium-scale systems such as PlaFRIM, which are numerous. Future work directions include testing their validity in larger scale systems, especially with larger file system deployments, and with other application access patterns, such as the file-per-process (N-N) strategy.

Author Contributions

Luan Teylo was in charge of software design, experimentation, data curation and visualization. All authors participated to the conceptualization of the work, methodology, analysis of the results and writing of the results. All authors read and approved the manuscript.

Acknowledgment

The authors would like to thank Julien Lelaurain and Brice Goglin for the support. All experiments were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr). This work was supported in part by the French National Research Agency (ANR) in the frame of DASH (ANR-17-CE25-0004), by the Project Région Nouvelle Aquitaine 2018-1R50119 "HPC scalable ecosystem" and by the "Adaptive multitier intelligent data manager for Exascale (ADMIRE)" project, funded by the European Union's Horizon 2020 JTI-EuroHPC Research and Innovation Programme (grant 956748).

References

[1] Io500. https://io500.org/list/sc19/io500.
[2] Lustre and io-500. https://www.opensfs.org/wp-content/uploads/2020/04/Lustre_IO500_v2.pdf. Accessed: 2022-01-14.
[3] Lustre® filesystem. https://www.lustre.org. Accessed: 2022-01-05.

[4] M Asch, T Moore, R Badia, M Beck, P Beckman, T Bidot, F Bodin, F Cappello, A Choudhary, B de Supinski, E Deelman, J Dongarra, A Dubey, G Fox, H Fu, S Girona, W Gropp, M Heroux, Y Ishikawa, K Keahey, D Keyes, W Kramer, J-F Lavignon, Y Lu, S Matsuoka, B Mohr, D Reed, S Requena, J Saltz, T Schulthess, R Stevens, M Swany, A Szalay, W Tang, G Varoquaux, J-P Vilotte, R Wisniewski, Z Xu, and I Zacharov. Big data and extreme-scale computing: Pathways to convergence-toward a shaping strategy for a future software and data ecosystem for scientific inquiry. *The International Journal of High Performance Computing Applications*, 32(4):435–479, 2018.

[5] BeeGFS. version 7.2.3. https://www.beegfs.io/, 2021.

[6] Babak Behzad, Surendra Byna, and Marc Snir. Optimizing i/o performance of hpc applications with autotuning. *ACM Transactions on Parallel Computing (TOPC)*, 5(4):1–27, 2019.

[7] IOR Benchmark. version 3.3.0. https://github.com/hpc/ior, 2021.

[8] Francieli Zanon Boito, Eduardo C Inacio, Jean Luca Bez, Philippe OA Navaux, Mario AR Dantas, and Yves Denneulin. A checkpoint of research on parallel i/o for high-performance computing. *ACM Computing Surveys (CSUR)*, 51(2):1–35, 2018.

[9] Jared Brzenski, Christopher Paolini, and Jose E. Castillo. Improving the i/o of large geophysical models using pnetcdf and beegfs. *Parallel Computing*, 104-105:102786, 2021.

[10] Zhen Cao, Vasily Tarasov, Hari Prasath Raman, Dean Hildebrand, and Erez Zadok. On the performance variation in modern storage stacks. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, pages 329–344, 2017.

[11] Hung Ching Chang, Bo Li, Matthew Grove, and Kirk W. Cameron. How processor speedups can slow down I/O Performance. In *Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS*, pages 395–404. IEEE, 2015.

[12] Fahim Chowdhury, Yue Zhu, Todd Heer, Saul Paredes, Adam Moody, Robin Goldstone, Kathryn Mohror, and Weikuan Yu. I/o characterization and performance evaluation of beegfs for deep learning. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP 2019, New York, NY, USA, 2019. Association for Computing Machinery.

[13] Matthieu Dorier, Gabriel Antoniu, Franck Cappello, Marc Snir, and Leigh Orf. Damaris: How to efficiently leverage multicore parallelism to achieve scalable, jitter-free I/O. In *Proceedings - 2012 IEEE International Conference on Cluster Computing, CLUSTER 2012*, pages 155–163, 2012.

[14] Ana Gainaru, Valentin Le Fèvre, and Guillaume Pallez. I/o scheduling strategy for periodic applications. *ACM Transactions on Parallel Computing*, 2019.

[15] Sunggon Kim, Alex Sim, Kesheng Wu, Suren Byna, Teng Wang, Yongseok Son, and Hyeonsang Eom. Dca-io: A dynamic i/o control scheme for parallel and distributed file systems. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 351–360. IEEE, 2019.

[16] B Lawrence, C Maynard, A Turner, X Guoc, and D Sloan-Murphyc. Parallel i/o performance benchmarking and investigation on multiple hpc architectures. Available at https://prace-ri.eu/wp-content/uploads/WP236.pdf (accessed 20 April 2022), 2017.

[17] Glenn Lockwood, Wucherl Yoo, Surendra Byna, N.J. Wright, Shane Snyder, Kevin Harms, Zachary Nault, and Philip Carns. Umami: a recipe for generating meaningful metrics through holistic i/o performance analysis. pages 55–60, 11 2017.

[18] Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert Latham, Robert Ross, Shane Snyder, and Stefan M. Wild. Analysis and correlation of application i/o performance and system-wide i/o activity. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–10, 2017.

[19] Nicholas Mills, F. Alex Feltus, and Walter B. Ligon III. Maximizing the performance of scientific data transfer by optimizing the interface between parallel file systems and advanced research networks. *Future Generation Computer Systems*, 79:190–198, 2018.

[20] Lucia Morganti, Elena Corni, Luca Lama, Carmelo Pellegrino, Francieli Zanon Boito, Ivan Merelli, Daniele D'Agostino, and Daniele Cesini. On low-power socs as storage bricks for bioinformatics. *Concurrency and Computation: Practice and Experience*, 32(10):e5415, 2020.

[21] Loïc Pottier, Rafael Ferreira da Silva, Henri Casanova, and Ewa Deelman. Modeling the performance of scientific workflow executions on hpc platforms with burst buffers. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 92–103, 2020.

[22] Hongzhang Shan and John Shalf. Using ior to analyze the i/o performance for hpc platforms. Technical report, Ernest Orlando Lawrence Berkeley NationalLaboratory, Berkeley, CA (US), 2007.

[23] Abdul Jabbar Saeed Tipu, Pádraig Conbhuí, and Enda Howley. Applying neural networks to predict hpc-i/o bandwidth over seismic data on lustre file system for exseisdat. *Cluster Computing*, 07 2021.

[24] Lipeng Wan, Matthew Wolf, Feiyi Wang, Jong Youl Choi, George Ostrouchov, and Scott Klasky. Comprehensive measurement and analysis of the user-perceived i/o performance in a production leadership-class storage system. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1022–1031, 2017.

[25] Feiyi Wang, Hyogi Sim, Cameron Harr, and Sarp Oral. Diving into petascale production file systems through large scale profiling and analysis. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, PDSW-DISCS '17, page 37–42, New York, NY, USA, 2017. Association for Computing Machinery.

[26] Teng Wang, Suren Byna, Glenn K. Lockwood, Shane Snyder, Philip Carns, Sunggon Kim, and Nicholas J. Wright. A zoom-in analysis of i/o logs to detect root causes of i/o performance bottlenecks. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 102–111, 2019.

[27] Teng Wang, Suren Byna, Glenn K. Lockwood, Shane Snyder, Philip Carns, Sunggon Kim, and Nicholas J. Wright. A zoom-in analysis of i/o logs to detect root causes of i/o performance bottlenecks. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 102–111, 2019.

[28] Teng Wang, Suren Byna, Glenn K Lockwood, Shane Snyder, Philip Carns, Sunggon Kim, and Nicholas J Wright. A zoom-in analysis of i/o logs to detect root causes of i/o performance bottlenecks. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 102–111. IEEE, 2019.

[29] Bing Xie, Zilong Tan, Philip Carns, Jeff Chase, Kevin Harms, Jay Lofstead, Sarp Oral, Sudharshan S. Vazhkudai, and Feiyi Wang. Applying machine learning to understand write performance of large-scale parallel filesystems. In *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*, pages 30–39, 2019.

[30] Cong Xu, Suren Byna, Vishwanath Venkatesan, Robert Sisneros, Omkar Kulkarni, Mohamad Chaarawi, and Kalyana Chadalavada. Lioprof: exposing lustre file system behavior for i/o middleware. In *2016 Cray User Group Meeting*, 2016.

[31] Bin Yang, Xu Ji, Xiaosong Ma, Xiyang Wang, Tianyu Zhang, Xiupeng Zhu, Nosayba El-Sayed, Haidong Lan, Yibo Yang, Jidong Zhai, Weiguo Liu, and Wei Xue. End-to-end i/o monitoring on a leading supercomputer. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, NSDI'19, page 379–394, USA, 2019. USENIX Association.

[32] Orcun Yildiz, Matthieu Dorier, Shadi Ibrahim, Rob Ross, and Gabriel Antoniu. On the root causes of cross-application i/o interference in hpc storage systems. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 750–759, 2016.

[33] Tiezhu Zhao and Jinlong Hu. Performance evaluation of parallel file system based on lustre and grey theory. In *2010 Ninth International Conference on Grid and Cloud Computing*, pages 118–123. IEEE, 2010.