 DOI 10.1109/PDSW51947.2020.00007

Towards On-Demand I/O Forwarding in HPC Platforms

Jean Luca Bez[1], Francieli Z. Boito[2], Alberto Miranda[3], Ramon Nou[3], Toni Cortes[3,4], Philippe O. A. Navaux[1]

[1]*Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS) — Porto Alegre, Brazil*
[2]*LaBRI, University of Bordeaux, Inria, CNRS, Bordeaux-INP – Bordeaux, France*
[3]*Barcelona Supercomputing Center (BSC) — Barcelona, Spain*
[4]*Polytechnic University of Catalonia — Barcelona, Spain*

{*jean.bez, navaux*}*@inf.ufrgs.br, francieli.zanon-boito@u-bordeaux.fr,*
{*alberto.miranda,ramon.nou*}*@bsc.es, toni@ac.upc.edu*

***Abstract*—I/O forwarding is an established and widely-adopted technique in HPC to reduce contention and improve I/O performance in the access to shared storage infrastructure. On such machines, this layer is often physically deployed on dedicated nodes, and their connection to the clients is static. Furthermore, the increasingly heterogeneous workloads entering HPC installations stress the I/O stack, requiring tuning and reconfiguration based on the applications' characteristics. Nonetheless, it is not always feasible in a production system to explore the potential benefits of this layer under different configurations without impacting clients. In this paper, we investigate the effects of I/O forwarding on performance by considering the application's I/O access patterns and system characteristics. We aim to explore when forwarding is the best choice for an application, how many I/O nodes it would benefit from, and whether not using forwarding at all might be the correct decision. To gather performance metrics, explore, and understand the impact of forwarding I/O requests of different access patterns, we implemented FORGE, a lightweight I/O forwarding layer in user-space. Using FORGE, we evaluated the optimal forwarding configurations for several access patterns on MareNostrum 4 (Spain) and Santos Dumont (Brazil) supercomputers. Our results demonstrate that shifting the focus from a static system-wide deployment to an on-demand reconfigurable I/O forwarding layer dictated by application demands can improve I/O performance on future machines.**

1. Introduction

Scientific applications impose ever-growing performance requirements to the High-Performance Computing (HPC) field. Increasingly heterogeneous workloads are entering HPC installations, from the traditionally compute-bound scientific simulations to Machine Learning applications and I/O bound Big Data workflows. While HPC clusters typically rely on a shared storage infrastructure powered by a Parallel File System (PFS) such as Lustre [1], GPFS [2], or Panasas [3], the increasing I/O demands of applications from fundamentally distinct domains stress this shared infrastructure. As systems grow in the number of compute nodes to accommodate larger applications and more concurrent jobs, the PFS is not able to keep providing performance due to increasing contention and interference [4], [5], [6].

I/O forwarding [7] is a technique that seeks to mitigate this contention by reducing the number of compute nodes concurrently accessing the PFS servers. Rather than having applications directly access the PFS, the forwarding technique defines a set of *I/O nodes* that are responsible for receiving application requests and forward them to the PFS, thus allowing the application of optimization techniques such as request scheduling and aggregation. Furthermore, I/O forwarding is file system agnostic and is transparently applied to applications. Due to these benefits, the technique is used by TOP500 machines [8], as showed in Table 1.

On such machines, I/O nodes are often physically deployed on special nodes with dedicated hardware, and their connection to the clients is static. Thus, a subset of compute nodes will only forward their requests to a single fixed I/O node. Typically, in this setup, all applications are forced to use forwarding when it is deployed in the machine.

In this paper, we seek to investigate the impact of I/O forwarding on performance, taking into account the application's I/O demands (i.e., their access patterns) and the overall system characteristics. We seek to explore when forwarding is the best choice (and how many I/O nodes an application would benefit from), and when not using it is the most suitable alternative. We hope that the design of the forwarding layer in future machines will build upon this knowledge, leading towards a more re-configurable I/O forwarding layer. The Sunway TaihuLight [9] and Tianhe-2A

TABLE 1. HIGHEST RAKED TOP 500 MACHINES THAT IMPLEMENT THE I/O FORWARDING TECHNIQUES (JUNE 2020).

Rank	Supercomputer	Compute Nodes	I/O Nodes
4	Sunway TaihuLight [9]	40,960	240
5	Tianhe-2A [6]	16,000	256
10	Piz Daint [10]	6,751	54
11	Trinity [11]	19,420	576

[6] already provide an initial support towards this plasticity.

Due to the physically static deployment of current I/O forwarding infrastructures, it is usually not possible to run experiments with varying numbers of I/O nodes. Furthermore, any reconfiguration of the forwarding layer typically requires acquiring, installing, and configuring new hardware. Thus, most machines are not easily re-configurable, and, in fact, end-users are not allowed to make modifications to this layer to prevent impacting a production system. We argue that a research/exploration alternative is required to enable obtaining an overview of the impact that different I/O access patterns might have under different I/O forwarding configurations. We seek a portable, simple-to-deploy solution, capable of covering different deployments and access patterns. Consequently, we implement a lightweight I/O forwarding layer named FORGE that can be deployed as a user-level job to gather performance metrics and aid in understanding the impact of forwarding in an HPC system. Our evaluation was conducted in two different supercomputers: the MareNostrum 4 at Barcelona Supercomputing Center (BSC), in Spain, and the Santos Dumont at the National Laboratory for Scientific Computation (LNCC), in Brazil. We demonstrate that the number of I/O nodes for 189 scenarios covering distinct access patterns is different. For 90.5%, using forwarding would indeed be the correct course of action. But using two I/O nodes, for instance, would only bring performance improvements for 44% of the scenarios.

The rest of this paper is organized as follows. Section 2 presents FORGE, our user-level implementation of the I/O forwarding technique. Our experimental methodology, the platforms and access patterns, are described in Section 3. We also present and discuss our results in that section. Finally, we conclude this paper in Section 5 and discuss future work.

2. FORGE: The I/O Forwarding Explorer

Existing I/O forwarding solutions in production today, such as IBM's CIOD [7] or Cray DVS [12] require dedicated hardware for I/O nodes and/or a restrictive software stack. They are, hence, not a straightforward solution to explore and evaluate I/O forwarding in machines that do not yet have this layer deployed or that may seek to modify existing deployments. Furthermore, modifications on deployment or tuning of these solutions would have a system-wide impact, even if just for exploration/testing purposes. Since many supercomputers such as the MareNostrum 4 and the Santos Dumont still do not have an I/O forwarding layer, we implemented such layer in user-space to allow sysadmins to evaluate the benefits and impact of using different number of I/O nodes under different workloads, without the need for production downtime or new hardware.

The goal of the **I/O Forwarding Explorer** (FORGE) is to evaluate new I/O optimizations (such as new request schedulers) as well as modifications on I/O forwarding deployment and configuration on large-scale clusters and supercomputers. Our solution is designed to be flexible enough to represent multiple I/O access patterns (number of processes, file layout, request spatiality, and request size) derived from Darshan [13]. As mentioned, it can be submitted to HPC queues as a normal user-space job to provide insights regarding the I/O node's behavior on the platform. FORGE is open-source and can be downloaded from https://github.com/jeanbez/forge.

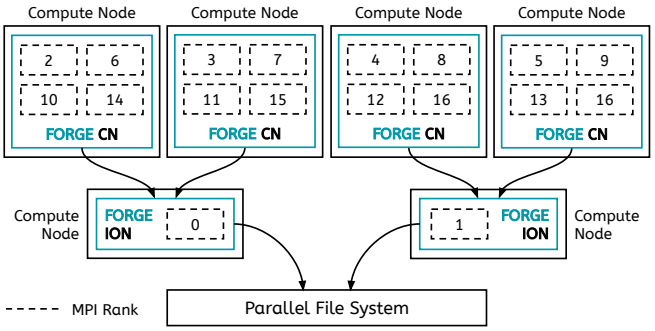


Figure 1. Overview of the architecture used by FORGE to implement the I/O forwarding technique at user-space. I/O nodes and computes nodes are distributed according to the hostfile.

We designed FORGE based on the general concepts applied in I/O forwarding tools such as IOFSL [14], IBM CIOD [7], and Cray DVS [12]. The common underlying idea present in these solutions is that clients emit I/O requests which are then intercepted and forwarded to one I/O node in the forwarding layer. This I/O node will receive requests from multiple processes of a given application (or even many applications), schedule those requests, and reshape the I/O flow by reordering and aggregating the requests to better suit the underlying PFS. The requests are then dispatched to the back-end PFS, which executes them and replies to the I/O node. Only then, the I/O node responds to the client (with the requested data, if that is the case) whether the operation was successful. I/O nodes typically have a pool of threads to handle multiple incoming requests and dispatch them to the PFS concurrently.

FORGE was built as an MPI application where M ranks are split into two subsets: compute nodes and I/O nodes. The first N processes act as *I/O forwarding servers*, and each should be placed exclusively on a compute node. The remaining $M - N$ processes act as an application, and are evenly distributed between the nodes, allocating more than one process per node if necessary, as depicted by Figure 1. The compute nodes issue I/O requests according to a user-defined JSON input file describing the I/O phases of an application. All I/O requests of a given compute node are sent by MPI messages to the defined I/O node. The operations are synchronous from the client's perspective, i.e., FORGE waits until the I/O operation to the PFS is complete to reply back to the client.

Each I/O node has a thread that listens to incoming requests and a pool of threads to dispatch these requests to the PFS and reply to the clients. Note that any metadata-related requests, such as those in `open()` and `close()` operations, are immediately executed. For `write()` and `read()` operations, the requests are placed on a queue of incoming requests that are scheduled using the AGIOS [15]

scheduling library, aggregated, and dispatched. AGIOS was chosen as it can be plugged into other forwarding solutions, and it contains several available schedulers. Regarding aggregation, in FORGE, requests to the same file and operation are handled in succession. Existing solutions, such as IOFSL, have a particular interface to issue list operations if the PFS supports it. We opted for a more general approach without relying on such feature being available.

3. Study of I/O Forwarding Performance

This section evaluates the impact of I/O forwarding on performance by observing the bandwidth achieved by different application' access patterns under different forwarding configurations. We seek to determine when forwarding is the best choice and how many I/O nodes an application would benefit from. Our experiments were conducted on MareNostrum 4[1] at Barcelona Supercomputing Center (BSC), Spain, and on Santos Dumont supercomputer at the National Laboratory for Scientific Computation (LNCC), Brazil.

MareNostrum 4 supercomputer uses $3,456$ Lenovo ThinkSystem SD530 compute nodes on 48 racks. Each node uses two Intel Xeon Platinum 8160 24C chips with 24 processors each at 2.1 GHz which totals to $165,888$ processes and 390 TB of main memory. Each node provides an Intel SSD DC S3520 Series with 240 GiB of available storage, usable within a job. A 100 Gb Intel Omni-Path Full-Fat Tree is used for the interconnection network and a total of 14 PB of storage capacity is offered by IBM's GPFS.

Santos Dumont has a total of $36,472$ CPU cores. The base system has 756 thin compute nodes with two Intel Xeon E5-2695v2 Ivy Bridge 2.40GHz 12-core processors, 64GB DDR3 RAM, and one 128GB SSD. There are three types of thin nodes (BullX B700 family), one fat node (BullX MESCA family), and one AI/ML/DL dedicated node (BullSequana X family). Its latest upgrade added 376 X1120 nodes with two Intel Xeon Cascade Lake Gold 6252 2.10GHz 24-core processors, and one 1TB SSD. 36 of these nodes have a 764GB DDR3 RAM, whereas the remainder have a 384GB DDR3 RAM. Compute nodes are connected to the LustreFS directly through a fat-tree non-blocking Infiniband FDR network. Each OSS has one OST made of 40 6TB HDD disks in a RAID6 (same for the MDS–MDT).

We explore FORGE with multiple access patterns and forwarding deployments in MareNostrum in Section 3.1. We expand our analysis using a subset of patterns in Santos Dumont in Section 3.2. We covered 189 scenarios, with at least 5 repetitions of each, considering the following factors:

- 8, 16, and 32 compute nodes;
- 12, 24, and 48 client processes per compute node (i.e., 96, 192, 384, 768, and 1536 processes);
- File layout: file-per-process or shared-file;
- Spatiality: contiguous or 1D-strided;
- Operation: writes with O_DIRECT enabled to account for caching effects present in the system;
- Request sizes of 32KB, 128KB, 512KB, 1MB, 4MB, 6MB, and 8MB synchronously issued until a given total size is transferred or a stonewall is reached.

As the compute nodes of both supercomputers' have 48 cores per node, we evaluate a scenario where an application uses all, half, or a quarter of its cores. Regarding file layout and spatiality, we opted for configurations commonly tested with benchmarks such as IOR [16] and MPI-IO Test [17].

TABLE 2. DESCRIPTION OF THE ACCESS PATTERNS WITH FORGE ON THE EXPERIMENT PRESENTED IN FIGURES 2 AND 4.

#	Nodes	Processes	File Layout	Request Spatiality	Request (KB)
A	32	1536	File-per-process	Contiguous	1024
B	32	1536	File-per-process	Contiguous	128
C	32	1536	Shared	Contiguous	1024
D	32	1536	Shared	Contiguous	4096
E	32	1536	Shared	1D-strided	512
F	16	192	Shared	Contiguous	32
G	16	192	Shared	1D-strided	128
H	8	192	File-per-process	Contiguous	8192
I	8	192	Shared	1D-strided	8192
J	16	384	Shared	Contiguous	128
K	16	384	Shared	Contiguous	8192
L	32	384	Shared	1D-strided	4096
M	32	384	Shared	1D-strided	512
N	8	384	Shared	Contiguous	4096
O	16	768	Shared	Contiguous	1024
P	32	768	Shared	Contiguous	1024
Q	16	768	Shared	1D-strided	1024
R	8	96	File-per-process	Contiguous	8192
S	8	96	Shared	1D-strided	6144
T	8	96	Shared	Contiguous	512

3.1. I/O Forwarding on MareNostrum 4

Figure 2 depicts the bandwidth measured at client-side, when multiple clients issue their requests following each access pattern and taking into account the number of available I/O nodes (0, 1, 2, 4, and 8). Each experiment was repeated at least 5 times, in random order, and spanning different days and periods of the day. Table 2 describes each depicted pattern. In Figure 2, only for scenario A directly accessing the PFS translates into higher I/O bandwidth. For L and M, the right choice would be to allocate one I/O node each. Patterns J and O would achieve higher bandwidth when four I/O nodes are given to the application. On the other hand, eight are required by scenarios B, G, H, and R. For the remaining scenarios, two I/O nodes is the ideal choice. The complete evaluation of the 189 experiments is available at https://doi.org/10.5281/zenodo.4016899.

For each one of 189 scenarios, it is possible to determine the different options an application (or access pattern) has to choose regarding how many I/O nodes it could use. For options that translate into similar achieved bandwidth, the smallest number of I/O nodes is the most reasonable choice. To verify how many choices we have for each pattern, we compute Dunn's test [18] for stochastic dominance, which reports the results among multiple pairwise comparisons

1. https://www.bsc.es/marenostrum/marenostrum

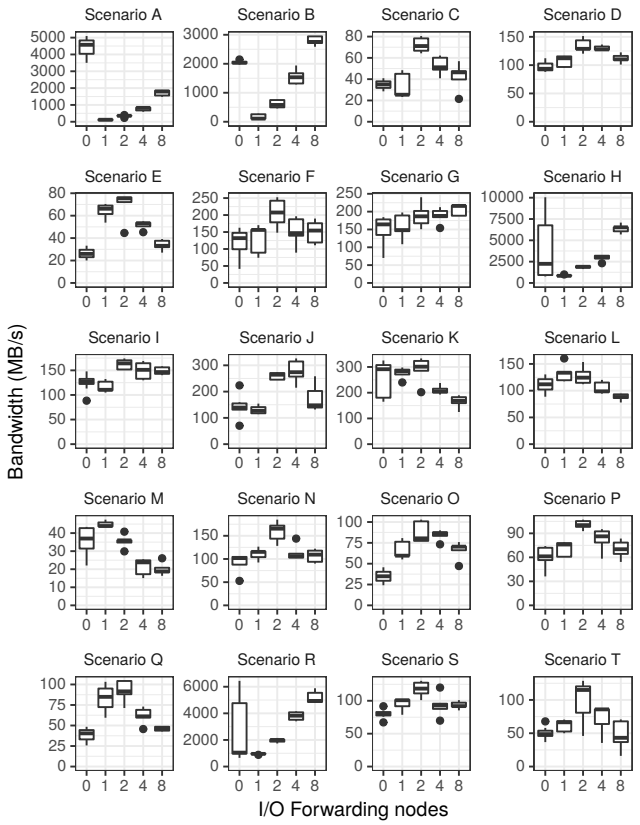


Figure 2. I/O bandwidth of distinct write access patterns and I/O nodes in the **MareNostrum 4** supercomputer. The y-axis is not the same.

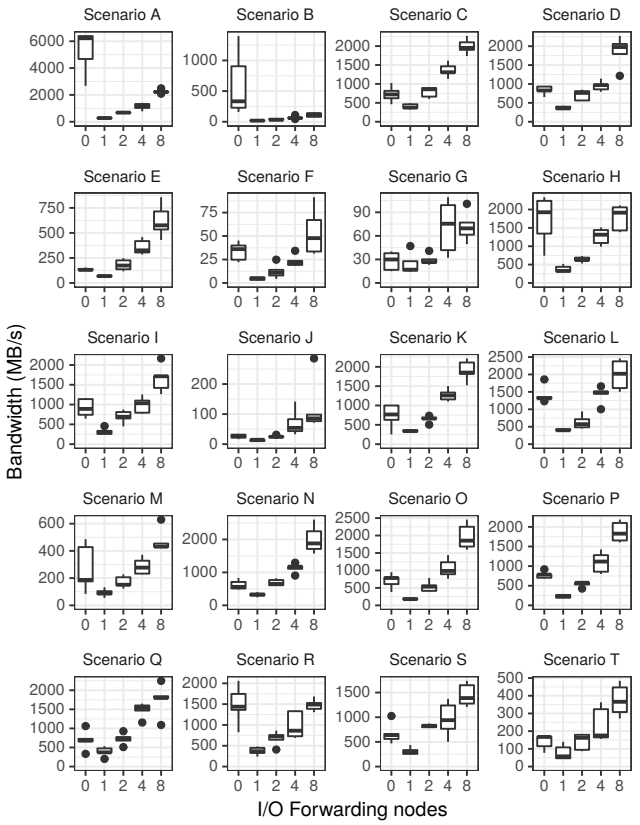
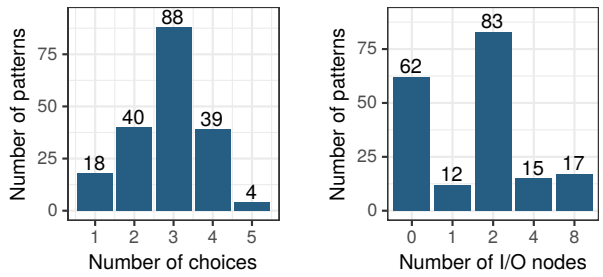


Figure 4. I/O bandwidth of distinct write access patterns and I/O nodes in the **Santos Dumont** supercomputer. The y-axis is not the same.



(a) (b)

Figure 3. (a) Number of choices each access pattern has which translates into statistically distinct I/O performance. (b) Access patterns grouped by the number of I/O nodes that would translates to the best performance.

after a Kruskal-Wallis [19] test for stochastic dominance among groups. Dunn is a non-parametric test that can be used to identify which specific means are significant from the others. The null hypothesis (H_0) for the test is that there is no difference between groups. The alternate hypothesis is that there is a difference. We used a level of significance of $\alpha = 0.05$, thus we reject H_0 if $p \leq alpha/2$.

Not using forwarding is always an option to be considered as it yields different bandwidth than when using forwarding, though it might not be the best choice for many cases. Figure 3(a) illustrates how many options an allocation policy would have to consider. For 88 ($\approx 46\%$) of the patterns, three possible choices would impact performance.

The optimal number of I/O nodes for each of the 189 scenarios, considering the available choices of I/O nodes, is different, as depicted by Figure 3(b). For 12 (6%), 83 (44%), 15 (8%), and 17 (9%) scenarios, the largest bandwidth is achieved by using 1, 2, 4, and 8 I/O nodes respectively. Whereas, for 62 scenarios (33%), not using forwarding is instead the best alternative. There does not seem to be a simple rule to fit all applications and system configurations, which is to be expected given the complexity of factors that can influence I/O performance. For instance, consider patterns A and B in MareNostrum. The size of requests makes not using forwarding the best for pattern A, whereas for B, using 8 I/O nodes translates more bandwidth. For pattern A and C, where the only difference is the file layout, C should use 2 I/O nodes. In Santos Dumont, 8 I/O nodes should be chosen for C instead of 0 as in A.

Considering the static physical setup of platforms where forwarding is present, applications often are not allowed to have direct access to the PFS other than by an I/O node. We can compare the maximum attained bandwidth one could achieve by forcibly using forwarding in all the cases (and the optimal number of I/O nodes) to not using this technique. For 90.5% (171 scenarios) using forwarding, for 9.5% (18 scenarios), directly accessing the PFS would instead increase

performance. Hence, both options should be available to applications to optimize I/O performance properly.

3.2. I/O Forwarding on Santos Dumont

We conducted a smaller evaluation on Santos Dumont (due to allocation restrictions) comprised of 20 patterns described in Table 2. Figure 4 depicts the I/O bandwidth of distinct write access patterns with a varying number of I/O nodes. One can notice that the forwarding layer setup's impact is not the same on MareNostrum and Santos Dumont.

A common behavior observed in Santos Dumont is that using more I/O nodes yields better performance. Regardless, the choice of using forwarding or not in this machine is still relevant. For instance, for the scenarios A, B, and H, direct access to the Lustre PFS translates into higher I/O bandwidth. Whereas, for some other scenarios, the benefits of using forwarding are perceived by the application after more than one I/O node is available. It is possible to notice that using a single I/O node is not the best choice for none of the tested patterns. If two were available, scenarios C, E, N, Q, and S would already see benefits. For D, G, I, J, K, L, M, O, P, and T, only when four I/O nodes are given to the application, in this machine, performance would increase. Other patterns, such as F and R, require eight I/O nodes instead to achieve the best performance. In practice, there will be an upper bound on how many of these resources are available to applications. This information could guide allocation policies to arbitrate the I/O nodes by allocating the resources to those that would achieve the highest bandwidth.

3.3. Discussion

As HPC systems tend to get more complex to accommodate new performance requirements and different applications, the forwarding layer might be re-purposed from a must-use to an on-demand approach. This layer has the potential to not only coordinate the access to the back-end PFS or avoid contention but also to transparently reshape the flow of I/O requests to be more suited to the characteristics and setup of the storage system. Our results demonstrate the intuitive notion that access patterns are impacted differently by using forwarding, and that the choice in the number of I/O nodes also plays an important role. Consequently, taking into account this information, novel forwarding mechanisms could benefit from this knowledge to allocate I/O nodes based on the demand. Our initial experiments with FORGE demonstrate that an idle compute node (or even a set of compute nodes) could be temporarily allocated to act as a forwarder and improve application and system performance.

Furthermore, as I/O nodes could be given only to applications that would benefit the most from them, interference on sharing these resources would be reduced or even eliminated. Instead, in today's setups, I/O nodes have to handle all requests from a subset of compute nodes, where concurrent applications (with very distinct characteristics) could be running. Yildiz et al. [4] demonstrate the impact of interference in the PFS, and Bez et al. [20] attested this when forwarding is also present. Moreover, Yu et al. [21] investigated the load imbalance on the forwarding nodes, where recruiting idle I/O nodes would be the solution. Instead, we argue that a shift of focus from the physical global deployment of this layer to the I/O demands of applications should instead guide this layer's future usage and distribution. Hence, I/O forwarding could be seen as an on-demand service for applications that would benefit from it, possibly recruiting temporary I/O nodes to avoiding interference present when sharing these resources.

4. Related Work

The I/O forwarding layer has been the focus of many research efforts [20], [22], [23] due to its potential to transparently improve applications' and systems' performance. Recently, studies have been motivated by the need of optimally allocating these I/O nodes in a cluster. Yu et al. [21] propose to recruit idle I/O nodes to mitigate the load imbalance problem of the I/O forwarding layer. The mapping of extra and primary I/O nodes is exclusive for an application. Consequently, that adds some flexibility to the default static solution. Ji et al. [24] propose a Dynamic Forwarding Resource Allocation (DRFA) which estimates the number of I/O nodes needed by a certain job based on its history records. DFRA remaps compute nodes to other than their default forwarding node assignments. They either grant more I/O nodes (for capacity) or unused I/O nodes (for isolation) based on the global application's behavior. Their strategy relies on resource over-provisioning and assumes that there are idle I/O to satisfy all allocations.

Conversely, we sought to expand such studies with an experimental analysis on how I/O forwarding would benefit performance on machines that do not have this layer physically deployed yet. Furthermore, we opted to explore with an access pattern granularity, targeting the baseline access patterns that describe an application's behavior to determine the number of I/O nodes that better suit each situation. The acquired knowledge can be used to guide allocation decisions for all applications that share the observed patterns.

Compared to the current state of the art, FORGE can assist system administrators in understanding how common access patterns behave under different forwarding setups. In its client-side, FORGE can be compared to IOR [16] or MPI-IO Test [17], where it is possible to describe various I/O workloads and run tests using the cluster's job queues. FORGE implements a forwarding layer's baseline concepts, including request forwarding, scheduling, and aggregation.

5. Conclusion

I/O forwarding is an established and widely-adopted technique in HPC to reduce contention and improve I/O performance in the access to shared storage infrastructure. However, it is not always possible to explore its advantages under different setups without impacting or disrupting production systems. This paper investigates I/O forwarding by

considering particular applications I/O access patterns and system configuration, rather than trying to guess a one-size-fits-all configuration for all applications. By determining when forwarding is the best choice for an application and how many I/O nodes it would benefit from, we can guide allocation policies and help reaching better decisions.

To understand the impact of forwarding I/O requests of different access patterns, we implemented FORGE, a lightweight forwarding layer in user-space. We explored 189 scenarios covering distinct access patterns and demonstrated that, as expected, the optimal number of I/O nodes varies depending on the application. While for 90.5% patterns forwarding would be the correct course of action, allocating two I/O nodes would only bring performance improvements for 44% of the scenarios. Our results on the MareNostrum and Santos Dumont supercomputers demonstrate that shifting the focus from a static system-wide deployment to an on-demand reconfigurable I/O forwarding layer dictated by application demands can improve I/O performance on future machines. Future work will harness this knowledge to guide and arbitrate on I/O nodes' allocation decisions. We seek to propose a production-ready solution capable of adding reallocation flexibility to this layer based on the application's access patterns and I/O demands.

Acknowledgments

This study was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. It has also received support from the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil; It is also partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grants PID2019-107255GB; and the Generalitat de Catalunya under contract 2014–SGR–1051. The author thankfully acknowledges the computer resources, technical expertise and assistance provided by the Barcelona Supercomputing Center - Centro Nacional de Supercomputación. The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer, which have contributed to the research results reported within this paper. URL: http://sdumont.lncc.br.

References

[1] SUN, "High-Performance Storage Architecture and Scalable Cluster File System," Sun Microsystems, Inc, Tech. Rep., 2007.

[2] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02. USA: USENIX Association, 2002, p. 19–es.

[3] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable Performance of the Panasas Parallel File System," in *6th USENIX Conference on File and Storage Technologies*, ser. FAST'08. USA: USENIX Association, 2008.

[4] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu, "On the Root Causes of Cross-Application I/O Interference in HPC Storage Systems," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Chicago, USA: IEEE, 2016, pp. 750–759.

[5] J. Yu, G. Liu, X. Li, W. Dong, and Q. Li, "Cross-layer coordination in the I/O software stack of extreme-scale systems," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 10, 2018.

[6] W. Xu, Y. Lu, Q. Li, E. Zhou, Z. Song, Y. Dong, W. Zhang, D. Wei, X. Zhang, H. Chen, J. Xing, and Y. Yuan, "Hybrid hierarchy storage system in MilkyWay-2 supercomputer," *Frontiers of Computer Science*, vol. 8, no. 3, pp. 367–377, 2014.

[7] G. Almási, R. Bellofatto, J. Brunheroto, C. Caşcaval, J. G. Castanos, L. Ceze *et al.*, "An overview of the Blue Gene/L system software organization," in *Euro-Par 2003 Parallel Processing*, Euro-Par 2003 Conference, Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 543–555.

[8] TOP500, *TOP500 List – June 2020*, 2020 (accessed September 1, 2020). [Online]. Available: https://www.top500.org/lists/2020/06/

[9] B. Yang, X. Ji, X. Ma, X. Wang, T. Zhang, X. Zhu, N. El-Sayed, H. Lan, Y. Yang, J. Zhai, W. Liu, and W. Xue, "End-to-end i/o monitoring on a leading supercomputer," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 379–394.

[10] S. Gorini, M. Chesi, and C. Ponti, "CSCS Site Update," http://opensfs.org/wp-content/uploads/2017/06/Wed11-GoriniStefano-LUG2017_20170601.pdf, 2017, [Online; Accessed 7-January-2020].

[11] A. for Computing at Extreme Scale Team, "Trinity Platform Introductioon and Usage Model," https://www.lanl.gov/projects/trinity/_assets/docs/trinity-usage-model-presentation.pdf, 2015, [Online; Accessed 7-January-2020].

[12] S. Sugiyama and D. Wallace, "Cray DVS: Data Virtualization Service," 2008.

[13] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," in *27th Symposium on Mass Storage Systems and Technologies (MSST)*, 2011, pp. 1–14.

[14] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan, "Scalable I/O forwarding framework for high-performance computing systems," in *2009 IEEE International Conference on Cluster Computing and Workshops*. New Orleans, LA, USA: IEEE, Aug 2009, pp. 1–10.

[15] F. Z. Boito, R. V. Kassick, P. O. A. Navaux, and Y. Denneulin, "AGIOS: Application-guided I/O scheduling for parallel file systems," in *2013 International Conference on Parallel and Distributed Systems*, International Conference on Parallel and Distributed Systems (ICPADS). Seoul, South Korea: IEEE, Dec 2013, pp. 43–50.

[16] HPC IO Repository, "IOR," https://github.com/hpc/ior, 2020.

[17] Los Alamos National Laboratory, "MPI-IO Test Bechmark," http://freshmeat.sourceforge.net/projects/mpiiotest, 2008.

[18] J. Dunn and O. J. Dunn, "Multiple comparisons among means," *American Statistical Association*, pp. 52–64, 1961.

[19] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[20] J. L. Bez, F. Z. Boito, L. M. Schnorr, P. O. A. Navaux, and J.-F. Méhaut, "TWINS: Server Access Coordination in the I/O Forwarding Layer," in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. St. Petersburg, Russia: IEEE, March 2017, pp. 116–123.

[21] J. Yu, G. Liu, W. Dong, X. Li, J. Zhang, and F. Sun, "On the load imbalance problem of I/O forwarding layer in HPC systems," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. Chengdu, China: IEEE, 2017, pp. 2424–2428.

[22] V. Vishwanath, M. Hereld, V. Morozov, and M. E. Papka, "Topology-Aware Data Movement and Staging for I/O Acceleration on Blue Gene/P Supercomputing Systems," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC'11. New York, NY, USA: ACM, 2011.

[23] F. Isaila, J. Blas, J. Carretero, R. Latham, and R. Ross, "Design and evaluation of multiple-level data staging for blue gene systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 946–959, June 2011.

[24] X. Ji, B. Yang, T. Zhang, X. Ma, X. Zhu, X. Wang, N. El-Sayed, J. Zhai, W. Liu, and W. Xue, "Automatic, Application-Aware I/O Forwarding Resource Allocation," in *Proceedings of the 17th USENIX Conference on File and Storage Technologies*, ser. FAST'19. USA: USENIX Association, 2019, pp. 265–279.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

Our experiments with FORGE, a lightweight I/O forwarding layer implemented in user-space, were conducted on MareNostrum 4 supercomputer at Barcelona Supercomputing Center (BSC), Spain, and on Santos Dumont supercomputer at the National Laboratory for Scientific Computation (LNCC), Brazil.

We covered 189 scenarios, with at least 5 repetitions of each, considering the following factors:

- 8, 16, and 32 compute nodes;
- 12, 24, and 48 client processes per compute node (i.e., 96, 192, 384, 768, and 1536 processes);
- File layout: file-per-process or shared-file;
- Spatiality: contiguous or 1D-strided;
- Operation: writes with O_DIRECT enabled to account for caching effects present in the system;
- Request sizes of 32KB, 128KB, 512KB, 1MB, 4MB, 6MB, and 8MB synchronously issued until a given total size is transferred or a stonewall is reached (in our experiments a stonewall of 60 seconds).

Details on how to compile and run FORGE are available in its repository at: https://doi.org/10.5281/zenodo.4016901.

The companion repository with experimental setup, results, plots, and analysis: https://doi.org/10.5281/zenodo.4016899.

FORGE needs the AGIOS scheduling library for the I/O nodes to have I/O request scheduling capabilities. To install AGIOS:

```
git clone https://gitlab.com/jeanbez/forge-agios agios
cd agios
make library
make library_install
```

You must also have the GSL - GNU Scientific Library installed:

```
apt install libgsl-dev
```

To build FORGE:

```
git clone https://github.com/jeanbez/forge
cd forge
mkdir build
cd build
cmake ..
make
```

Before using FORGE to explore I/O forwarding in a cluster or supercomputer, you first need to configure the AGIOS scheduling library and then the scenario (setup and configuration) you want to explore. You need to copy some files to `/tmp` on each node AGIOS will run. These files are required as they contain configuration parameters and access times. More information about these files, please refer to the AGIOS repository and paper.

```
cd forge
cp agios/* /tmp/
```

FORGE receives a JSON file with the parameters for the execution. An example of a configuration file is presented below. Please refer to FORGE's repository for a list of all parameters and accepted values.

```
{
    "forwarders": "1",
    "handlers": "16",
    "dispatchers": "16",
    "path": "/mnt/pfs/test",
    "number_of_files": "shared",
    "spatiality": "contiguous",
    "total_size": "4294967296",
    "request_size": "32768",
    "validation": "0"
}
```

Furthermore, FORGE expects a hostfile with proper mapping of processes per compute node. Since the first *N* MPI processes will act as forwarders, you need to ensure that the first *N* nodes in the list have a single slot available.

Once you have the configuration file prepared, you can launch FORGE. However, notice that you need to start additional forwarders MPI processes. For instance, if you want to run with 128 clients and 4 forwarders, you need to use `--op 132`. The first forwarders MPI processes will be placed in separate nodes (one per node if your hostfile was correctly defined). The remainder of the process will be allocated to other compute nodes.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: All author-created data artifacts are maintained in a public repository under an OSI-approved license.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

Author-Created or Modified Artifacts:

```
Persistent ID: https://doi.org/10.5281/zenodo.4016901
Artifact name: FORGE Source Code Repository

Persistent ID: https://doi.org/10.5281/zenodo.4016899
Artifact name: Data Repository
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: The MareNostrum 4 supercomputer (MN4) uses 3,456 Lenovo ThinkSystem SD530 compute nodes on 48 racks. Each node uses two Intel Xeon Platinum 8160 24C chips with 24 processors each at 2.1 GHz which totals to 165,888 processes and 390 TB of main memory. Each node provides an Intel SSD DC S3520

Series with 240 GiB of available storage, usable within a job. A 100 Gb Intel Omni-Path Full-Fat Tree is used for the interconnection network and a total of 14 PB of storage capacity is offered by IBM's GPFS. The Santos Dumont supercomputer (SDumont) has a total of 36,472 CPU cores. The base system has 756 thin compute nodes with two Intel Xeon E5-2695v2 Ivy Bridge 2.40GHz 12-core processors, 64GB DDR3 RAM, and one 128GB SSD. There are three types of thin nodes (BullX B700 family), one fat node (BullX MESCA family), and one AI/ML/DL dedicated node (BullSequana X family). Its latest upgrade added 376 X1120 nodes with two Intel Xeon Cascade Lake Gold 6252 2.10GHz 24-core processors, and one 1TB SSD. 36 of these nodes have a 764GB DDR3 RAM, whereas the remainder have a 384GB DDR3 RAM. Compute nodes are connected to the LustreFS directly through a fat-tree non-blocking Infiniband FDR network. Each OSS has one OST made of 40 6TB HDD disks in a RAID6 (same for the MDS-MDT).

Operating systems and versions: MN4 nodes use the Red Hat Enterprise Linux Server 7.5 (Maipo). SDumont nodes use RedHat Linux 7.6 and the Bull Supercomputing Cluster Suite 5 Software Stack.

Compilers and versions: GCC 8.1.0 (MN4) and GCC 8.3 (SDumont).

Libraries and versions: Open MPI 4.0.2 (MN4) and GNU Scientific Library 2.4 (MN4). Open MPI 4.0.4 UCX (SDumont) and GNU Scientific Library 2.6 (SDumont).