GekkoFS – A temporary distributed file system for HPC applications

Short paper

Marc-André Vef*, Nafiseh Moti*, Tim Süß*, Tommaso Tocci†,
Ramon Nou†, Alberto Miranda†, Toni Cortes†§, André Brinkmann*
* Johannes Gutenberg University Mainz, Mainz, Germany
† Barcelona Supercomputing Center, Barcelona, Spain § Universitat Politecnica de Catalunya, Barcelona, Spain

***Abstract*—We present *GekkoFS*, a temporary, highly-scalable burst buffer file system which has been specifically optimized for new access patterns of data-intensive High-Performance Computing (HPC) applications. The file system provides relaxed POSIX semantics, only offering features which are actually required by most (not all) applications. It is able to provide scalable I/O performance and reaches millions of metadata operations already for a small number of nodes, significantly outperforming the capabilities of general-purpose parallel file systems.**

***Index Terms*—Distributed File Systems, HPC, Burst Buffers**

I. Introduction

High-Performance Computing (HPC) applications are significantly changing. Traditional HPC applications have been compute-bound, large-scale simulations, while today's HPC community is additionally moving towards the generation, processing, and analysis of massive amounts of experimental data. This trend, known as *data-driven science*, is affecting many different scientific fields, some of which have made significant progress tackling previously unaddressable challenges thanks to newly developed techniques [15], [30].

Most data-driven workloads are based on new algorithms and data structures like graph databases which impose new requirements on HPC file systems [22], [39]. They include, e.g., large numbers of metadata operations, data synchronization, non-contiguous and random access patterns, and small I/O requests [9], [22]. Such operations differ significantly from past workloads which mostly performed sequential I/O operations on large files. They do not only slow down data-driven applications themselves but can also heavily disrupt other applications that are concurrently accessing the shared storage system [11], [35]. Consequently, traditional parallel file systems (PFS) cannot handle these workloads efficiently and data-driven applications suffer from prolonged I/O latencies, reduced throughput, and long waiting times.

Software-based approaches, e.g., application modifications or middleware and high-level libraries [12], [19], try to support data-driven applications to align the new access patterns to the capabilities of the underlying PFS. Yet, adapting such software is typically time-consuming, difficult to couple with big data and machine learning libraries, or sometimes (based on the underlying algorithms) just impossible.

Hardware-based approaches move from magnetic disks, the main backend technology for PFSs, to NAND-based solid-state drives (SSDs). Nowadays, many supercomputers deploy SSDs which can be used as dedicated burst buffers [18] or as *node-local* burst buffers. To achieve high metadata performance, they can be deployed in combination with a dynamic *burst buffer file system* [3], [40].

Generally, burst buffer file systems increase performance compared to a PFS without modifying an application. Therefore, they typically support POSIX which provides the standard semantics accepted by most application developers. Nevertheless, enforcing POSIX can severely reduce a PFS' peak performance [38]. Further, many POSIX features are not required for most scientific applications [17], especially if they can exclusively access the file system. Similar argumentations hold for other advanced features like fault tolerance or security.

In this work, we present *GekkoFS*, a temporarily deployed, highly-scalable distributed file system for HPC applications which aims to accelerate I/O operations of common HPC workloads that are challenging for modern PFSs. GekkoFS pools together fast node-local storage resources and provides a global namespace accessible by all participating nodes. It relaxes POSIX by removing some of the semantics that most impair I/O performance in a distributed context and takes previous studies on the behavior of HPC applications into account [17] to optimize the most used file system operations.

For load-balancing, all data and metadata are distributed across all nodes using the HPC RPC framework *Mercury* [34]. The file system runs in user-space and can be easily deployed in under 20 seconds on a 512 node cluster by any user. Therefore, it can be used in a number of temporary scenarios, e.g., during the lifetime of a compute job or in longer-term use cases, e.g., campaigns. We demonstrate how our lightweight, yet highly distributed file system GekkoFS reaches scalable data and metadata performance with tens of millions of metadata operations per second on a 512 node cluster while still providing strong consistency for file system operations that target a specific file or directory.

II. Related work

General-purpose PFSs like GPFS, Lustre, BeeGFS, or PVFS [4], [14], [27], [31], [32] provide long-term storage which is

 DOI 10.1109/CLUSTER.2018.00049

mostly based on magnetic disks. GekkoFS instead builds a short-term, separate namespace from fast node-local SSDs that is only temporarily accessible during the runtime of a job or a campaign. As such, GekkoFS can be categorized into the class of *node-local burst buffer* file systems, while *remote-shared burst buffer* file systems use dedicated, centralized I/O nodes [40], e.g., DDN's IME [1].

In general, node-local burst buffers are fast, intermediate storage systems that aim to reduce the PFS' load and the applications' I/O overhead [18]. They are typically collocated with nodes running a compute job, but they can also be dependent on the backend PFS [3] or in some cases even directly managed by it [24]. BurstFS [40], perhaps the most related work to ours, is a standalone burst buffer file system, but, unlike GekkoFS, is limited to write data locally. BeeOND [14] can create a job-temporal file system on a number of nodes similar to GekkoFS. However, in contrast to our file system, it is POSIX compliant and our measurements show a much higher metadata throughput than offered by BeeOND [36].

The management of inodes and related directory blocks are the main scalability limitations of file systems in a distributed environment. Typically, general-purpose PFSs distribute data across all available storage targets. As this technique works well for data, it does not achieve the same throughput when handling metadata [5], [28], although the file system community presented various techniques to tackle this challenge [3], [13], [25], [26], [41], [42]. The performance limitation can be attributed to the sequentialization enforced by underlying POSIX semantics which is particularly degrading throughput when a huge number of files is created in a single directory from multiple processes. This workload, common to HPC environments [3], [24], [25], [37], can become an even bigger challenge for upcoming data-science applications. GekkoFS is built on a new technique to handle directories and replaces directory entries by objects, stored within a strongly consistent key-value store which helps to achieve tens of millions of metadata operations for billions of files.

III. Design and Implementation

GekkoFS offers a user-space file system for the lifetime of a particular use case, e.g., within the context of an HPC job. The file system uses the available local storage of compute nodes to distribute data and metadata and combines their node-local storage into a single global namespace.

The file system's main goal focuses on scalability and consistency. It should therefore scale to an arbitrary number of nodes to benefit from current and future storage and network technologies. Further, GekkoFS should provide the same consistency as POSIX for file system operations that access a specific data file. However, consistency of directory operations, for instance, can be relaxed. Finally, GekkoFS should be hardware independent to efficiently use today's network technologies as well as any modern and future storage hardware that is accessible by the user.

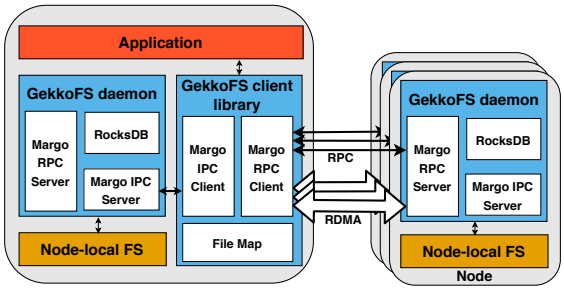


Fig. 1: GekkoFS architecture

A. POSIX relaxation

Similarly to PVFS [6] and OrangeFS [21], GekkoFS does not provide complex global locking mechanisms. In this sense, applications should be responsible to ensure that no conflicts occur, in particular, w.r.t. overlapping file regions. However, the lack of distributed locking has consequences for operations where the number of affected file system objects is unknown a priori, such as `readdir()` called by the `ls -l` command. In these *indirect file system operations*, GekkoFS does not guarantee to return the current state of the directory and follows the eventual-consistency model. Furthermore, each file system operation is synchronous without any form of caching to reduce file system complexity and to allow for an evaluation of its raw performance capabilities.

GekkoFS does not support move or rename operations or linking functionality as HPC application studies have shown that these features are rarely or not used at all during the execution of a parallel job [17]. Finally, security management in the form of access permissions is not maintained by GekkoFS since it already implicitly follows the security protocols of the node-local file system.

B. Architecture

GekkoFS' architecture (see Figure 1) consists of two main components: a client library and a server process. An application that uses GekkoFS must first preload the client interposition library which intercepts all file system operations and forwards them to a server (*GekkoFS daemon*), if necessary. The GekkoFS daemon, which runs on each file system node, receives forwarded file system operations from clients and processes them independently, sending a response when finished. In the following paragraphs, we describe the client and daemon in more detail.

a) GekkoFS client: The client consists of three components: 1) An interception interface that catches relevant calls to GekkoFS and forwards unrelated calls to the node-local file system; 2) a file map that manages the file descriptors of open files and directories, independently of the kernel; and 3) an RPC-based communication layer that forwards file system requests to local/remote GekkoFS daemons.

Each file system operation is forwarded via an RPC message to a specific daemon (determined by hashing of the file's path) where it is directly executed. In other words, GekkoFS

uses a pseudo-random distribution to spread data and metadata across all nodes, also known as *wide-striping*. Because each client is able to independently resolve the responsible node for a file system operation, GekkoFS does not require central data structures that keep track of where metadata or data is located. To achieve a balanced data distribution for large files, data requests are split into equally sized chunks before they are distributed across file system nodes. If supported by the underlying network fabric protocol, the client exposes the relevant chunk memory region to the daemon, accessed via *remote-direct-memory-access* (RDMA).

b) GekkoFS daemon: GekkoFS daemons consist of three parts: 1) A key-value store (KV store) used for storing metadata; 2) an I/O persistence layer that reads/writes data from/to the underlying local storage system (one file per chunk); and 3) an RPC-based communication layer that accepts local and remote connections to handle file system operations.

Each daemon operates a single local RocksDB KV store [10]. RocksDB is optimized for NAND storage technologies with low latencies and fits GekkoFS' needs as SSDs are primarily used as node-local storage in today's HPC clusters.

For the communication layer, we leverage on the *Mercury* RPC framework [34]. It allows GekkoFS to be network-independent and to efficiently transfer large data within the file system. Within GekkoFS, Mercury is interfaced indirectly through the *Margo* library which provides *Argobots*-aware wrappers to Mercury's API with the goal to provide a simple multi-threaded execution model [7], [33]. Using Margo allows GekkoFS daemons to minimize resource consumption of Margo's progress threads and handlers which accept and handle RPC requests [7].

IV. Evaluation

We evaluated the performance of GekkoFS based on various unmodified microbenchmarks which catch access patterns that are common in HPC applications. Our experiments were conducted on the *MOGON II* supercomputer, located at the Johannes Gutenberg University Mainz in Germany. All experiments were performed on Intel 2630v4 Intel Broadwell processors (two sockets each). The main memory capacity inside the nodes ranges from 64 GiB up to 512 GiB of memory. MOGON II uses 100 Gbit/s Intel Omni-Path to establish a fat-tree network between all compute nodes. In addition, each node provides a data center Intel SATA SSD DC S3700 Series as scratch-space (*XFS* formatted) usable within a compute job. We used these SSDs for storing data and metadata of GekkoFS which uses an internal chunk size of 512 KiB.

Before each experiment iteration, GekkoFS daemons are restarted (requiring less than 20 seconds for 512 nodes), all SSD contents are removed, and kernel buffer, inode, and dentry caches are flushed. The GekkoFS daemon and the application under test are pinned to separate processor sockets to ensure that file system and application do not interfere with each other.

A. Metadata performance

We simulated common metadata intensive HPC workloads using the unmodified *mdtest* microbenchmark [20] to evaluate GekkoFS' metadata performance and compare it against a Lustre parallel file system. Although GekkoFS and Lustre have different goals, we point out the performances that can be gained by using GekkoFS as a burst buffer file system. In our experiments, mdtest performs *create*, *stat*, and *remove* operations in parallel in a single directory – an important workload in many HPC applications and among the most difficult workloads for a general-purpose PFS [37].

Each operation on GekkoFS was performed using 100,000 zero-byte files per process (16 processes per node). From the user application's perspective, all created files are stored within a single directory. However, due to GekkoFS' internally kept flat namespace, there is conceptually no difference in which directory files are created. This is in contrast to a traditional PFS that may perform better if the workload is distributed among many directories instead of in a single directory. Figure 2 compares GekkoFS with Lustre in three scenarios with up to 512 nodes: file creation, file stat, and file removal. The y-axis depicts the corresponding operations per second that were achieved for a particular workload on a logarithmic scale. Each experiment was run at least five times with each data point representing the mean of all iterations. GekkoFS' workload scaled with 100,000 files per process, while Lustre's workload was fixed to four million files for all experiments. We fixed the number of files for Lustre's metadata experiments because Lustre was otherwise detecting hanging nodes when scaling to too many files.

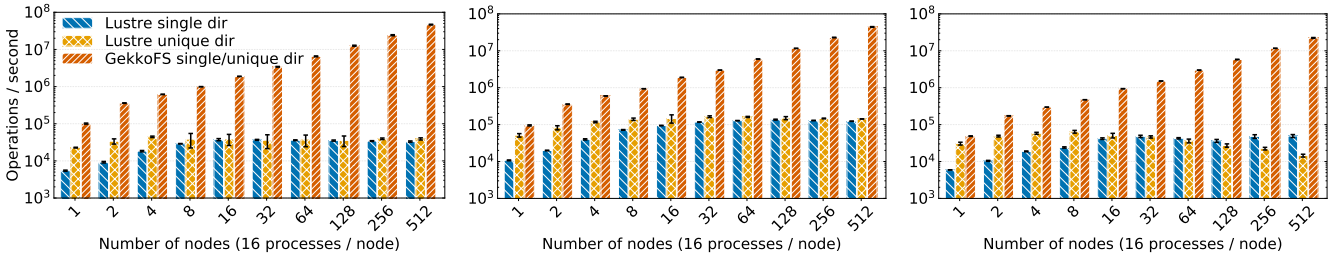
Lustre experiments were run in two configurations: All processes operated in a single directory (`single dir`) or each process worked in its own directory (`unique dir`). Moreover, Lustre's metadata performance was evaluated while the system was accessible by other applications as well.

As seen in Figure 2, GekkoFS outperforms Lustre by a large margin in all scenarios and shows close to linear scaling, regardless of whether Lustre processes operated in a single or in an isolated directory. Compared to Lustre, GekkoFS achieved around 46 million creates/s (~1,405x), 44 million stats/s (~359x), and 22 million removes/s (~453x) at 512 nodes. The standard deviation was less than 3.5% which was computed as the percentage of the mean.

B. Data performance

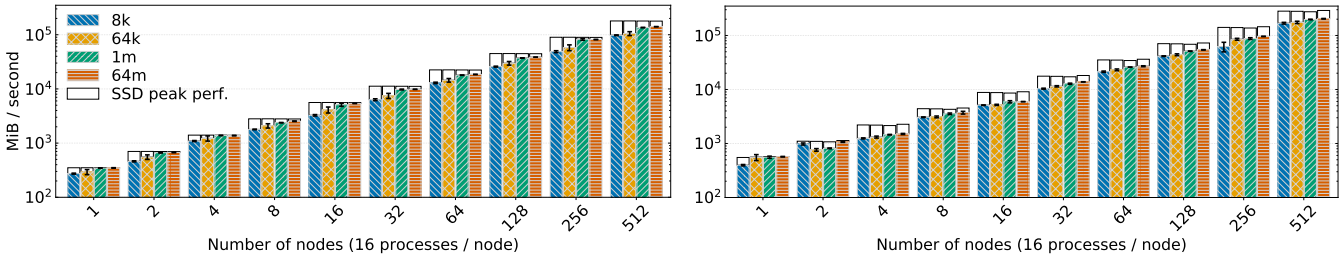
We used the unmodified *IOR* [20] microbenchmark to evaluate GekkoFS' I/O performance for sequential and random access patterns in two scenarios: Each process is accessing its own file (file-per-process) and all processes access a single file (shared file). We used 8 KiB, 64 KiB, 1 MiB, and 64 MiB *transfer sizes* to assess the performances for many small I/O accesses and for few large I/O requests. We ran 16 processes on each client, each process writing and reading 4 GiB in total.

GekkoFS data performance is not compared with the Lustre scratch file system as the peak performance of the used Lustre partition, around 12 GiB/s, is already reached for $\leq$ 10 nodes



(a) Create throughput (b) Stat throughput (c) Remove throughput

Fig. 2: GekkoFS' file create, stat, and remove throughput for an increasing number of nodes compared to a Lustre file system.



(a) Write throughput (b) Read throughput

Fig. 3: GekkoFS' sequential throughput for each process operating on its own file compared to the plain SSD peak throughput.

for sequential I/O patterns. Moreover, Lustre has shown to scale linearly in larger deployments with more OSSs and OSTs being available [23].

Figure 3 shows GekkoFS' sequential I/O throughput in MiB/s, representing the mean of at least five iterations, for an increasing number of nodes for different transfer sizes. In addition, each data point is compared to the peak performance that all aggregated SSDs could deliver for a given node configuration, visualized as a white rectangle, indicating GekkoFS' SSD usage efficiency. In general, every result demonstrates GekkoFS' close to linear scalability, achieving about 141 GiB/s (~80% of the aggregated SSD peak bandwidth) and 204 GiB/s (~70% of the aggregated SSD peak bandwidth) for write and read operations for a transfer size of 64 MiB for 512 nodes. At 512 nodes, this translates to more than 13 million write IOPS and more than 22 million read IOPS, while the average latency can be bounded by at most 700 μs for file system operations with a transfer size of 8 KiB.

For the file-per-process cases, sequential and random access I/O throughput are similar for transfer sizes larger than the file system's chunk size. This is due to transfer sizes larger than the chunk size internally access whole chunk files while smaller transfer sizes access one chunk at a random offset. Consequently, random accesses for large transfer sizes are conceptually the same as sequential accesses. For smaller transfer sizes, e.g., 8 KiB, random write and read throughput decreased by approximately 33% and 60%, respectively, for 512 nodes owing to the resulting random access to positions within the chunks.

For the shared file cases, a drawback of GekkoFS' synchronous and cache-less design becomes visible. No more than approximately 150K write operations per second were achieved. This was due to network contention on the daemon which maintains the shared file's metadata whose size needs to be constantly updated. To overcome this limitation, we added a rudimentary client cache to locally buffer size updates of a number of write operations before they are send to the node that manages the file's metadata. As a result, shared file I/O throughput for sequential and random access were similar to file-per-process performances since chunk management on the daemon is then conceptually indifferent in both cases.

V. Conclusion and Acknowledgements

We have introduced and evaluated GekkoFS, a new burst buffer file system for HPC applications with relaxed POSIX-semantics, allowing it to achieve millions of metadata operations even for a small number of nodes and close to linear scalability in various data and metadata use cases. Next, we plan to extend GekkoFS in three directions: Investigate GekkoFS' with various chunk sizes, evaluate benefits of caching, and explore different data distribution patterns.

The work has been funded by the German Research Foundation (DFG) through the ADA-FS project as part of the Priority Programme 1648. It is also supported by the Spanish Ministry of Science and Innovation (TIN2015–65316), the Generalitat de Catalunya (2014–SGR–1051), as well as the European Union's Horizon 2020 Research and Innovation Programme (NEXTGenIO, 671951) and the European Comission's BigStorage project (H2020-MSCA-ITN-2014-642963). This research was conducted using the supercomputer MOGON II and services offered by the Johannes Gutenberg University Mainz.

REFERENCES

[1] Infinite Memory Engine. https://www.ddn.com/products/ime-flash-native-data-cache.

[2] The Open Group Base Specifications Issue 7(IEEE Std 1003.1-2008). http://pubs.opengroup.org/onlinepubs/9699919799/.

[3] J. Bent, G. A. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, "PLFS: a checkpoint filesystem for parallel applications," in *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC), November 14-20, Portland, Oregon, USA*, 2009.

[4] P. J. Braam and P. Schwan, "Lustre: The intergalactic file system," in *Ottawa Linux Symposium*, 2002, p. 50.

[5] P. Carns, Y. Yao, K. Harms, R. Latham, R. Ross, and K. Antypas, "Production i/o characterization on the cray xe6," in *Proceedings of the Cray User Group meeting*, vol. 2013, 2013.

[6] P. H. Carns, W. B. L. III, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for linux clusters," in *4th Annual Linux Showcase & Conference 2000, Atlanta, Georgia, USA, October 10-14, 2000*, 2000.

[7] P. H. Carns, J. Jenkins, C. D. Cranor, S. Atchley, S. Seo, S. Snyder, and R. B. Ross, "Enabling NVM for data-intensive scientific services," in *4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads, INFLOW@OSDI 2016, Savannah, GA, USA, November 1, 2016.*, 2016.

[8] A. Choudhary, W.-k. Liao, K. Gao, A. Nisar, R. Ross, R. Thakur, and R. Latham, "Scalable i/o and analytics," in *Journal of Physics: Conference Series*, vol. 180, no. 1, 2009, p. 012048.

[9] P. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed, "Input/output characteristics of scalable parallel applications," in *Proceedings Supercomputing '95, San Diego, CA, USA, December 4-8, 1995*, 1995, p. 59.

[10] S. Dong, M. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum, "Optimizing space amplification in rocksdb," in *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017.

[11] M. Dorier, G. Antoniu, R. B. Ross, D. Kimpe, and S. Ibrahim, "Calciom: Mitigating I/O interference in HPC systems through cross-application coordination," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23, 2014*, 2014, pp. 155–164.

[12] M. Folk, A. Cheng, and K. Yates, "Hdf5: A file format and i/o library for high performance computing applications," in *Proceedings of supercomputing*, vol. 99, 1999, pp. 5–33.

[13] W. Frings, F. Wolf, and V. Petkov, "Scalable massively parallel I/O to task-local files," in *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC), November 14-20, Portland, Oregon, USA*, 2009.

[14] F. Herold, S. Breuner, and J. Heichler, "An introduction to beegfs," 2014, https://www.beegfs.io/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf.

[15] T. Hey, S. Tansley, and K. M. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[16] R. Latham, R. B. Ross, and R. Thakur, "The impact of file systems on MPI-IO scalability," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 19-22, 2004, Proceedings*, 2004, pp. 87–96.

[17] P. H. Lensing, T. Cortes, J. Hughes, and A. Brinkmann, "File system scalability with highly decentralized metadata on independent storage devices," in *IEEE/ACM 16th International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, Colombia, May 16-19*, 2016, pp. 366–375.

[18] N. Liu, J. Cope, P. H. Carns, C. D. Carothers, R. B. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST 2012, April 16-20, 2012, Asilomar Conference Grounds, Pacific Grove, CA, USA*, 2012, pp. 1–11.

[19] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," in *6th International Workshop on Challenges of Large Applications in Distributed Environments, CLADE@HPDC 2008, Boston, MA, USA, June 23, 2008*, 2008, pp. 15–24.

[20] "Mdtest metadata benchmark and ior data benchmark," 2018, https://github.com/hpc/ior.

[21] M. Moore, D. Bonnie, B. Ligon, M. Marshall, W. Ligon, N. Mills, E. Quarles, S. Sampson, S. Yang, and B. Wilson, "Orangefs: Advancing pvfs," *FAST poster session*, 2011.

[22] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. L. Best, "File-access characteristics of parallel scientific workloads," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 10, pp. 1075–1089, 1996.

[23] S. Oral, D. A. Dillow, D. Fuller, J. Hill, D. Leverman, S. S. Vazhkudai, F. Wang, Y. K. , J. Rogers, J. James Simmons, and R. Miller, "Olcfs 1 tb/s, next-generation lustre file system," in *Proceedings of Cray User Group Conference (CUG 2013)*, 2013.

[24] S. Oral and G. Shah, "Spectrum scale enhancements for coral. presentation slides at supercomputing'16." 2016, http://files.gpfsug.org/presentations/2016/SC16/11_Sarp_Oral_Gautam_Shah_Spectrum_Scale_Enhancements_for_CORAL_v2.pdf.

[25] S. Patil and G. A. Gibson, "Scale and concurrency of GIGA+: file system directories with millions of files," in *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011*, 2011, pp. 177–190.

[26] S. Patil, K. Ren, and G. Gibson, "A case for scaling HPC metadata performance through de-specialization," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, November 10-16, 2012*, 2012, pp. 30–35.

[27] Y. Qian, X. Li, S. Ihara, L. Zeng, J. Kaiser, T. Süß, and A. Brinkmann, "A configurable rule based classful token bucket filter network request scheduler for the lustre file system," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, CO, USA, November 12 - 17*, 2017, pp. 6:1–6:12.

[28] K. Ren, Q. Zheng, S. Patil, and G. A. Gibson, "Indexfs: Scaling file system metadata performance with stateless caching and bulk insertion," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, LA, USA, November 16-21, 2014*, 2014, pp. 237–248.

[29] D. Ritchie and K. Thompson, "The UNIX time-sharing system (reprint)," *Commun. ACM*, vol. 26, no. 1, pp. 84–89, 1983.

[30] R. Ross, R. Thakur, and A. Choudhary, "Achievements and challenges for i/o in computational science," in *Journal of Physics: Conference Series*, vol. 16, no. 1, 2005, p. 501.

[31] R. B. Ross and R. Latham, "PVFS - PVFS: a parallel file system," in *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, November 11-17, 2006, Tampa, FL, USA*, 2006, p. 34.

[32] F. B. Schmuck and R. L. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proceedings of the FAST '02 Conference on File and Storage Technologies, January 28-30, Monterey, California, USA*, 2002, pp. 231–244.

[33] S. Seo, A. Amer, P. Balaji, C. Bordage, G. Bosilca, A. Brooks, P. H. Carns, A. Castelló, D. Genet, T. Hérault, S. Iwasaki, P. Jindal, L. V. Kalé, S. Krishnamoorthy, J. Lifflander, H. Lu, E. Meneses, M. Snir, Y. Sun, K. Taura, and P. H. Beckman, "Argobots: A lightweight low-level threading and tasking framework," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 3, pp. 512–526, 2018.

[34] J. Soumagne, D. Kimpe, J. A. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. B. Ross, "Mercury: Enabling remote procedure call for high-performance computing," in *2013 IEEE International Conference on Cluster Computing, CLUSTER 2013, Indianapolis, IN, USA, September 23-27, 2013*, 2013, pp. 1–8.

[35] S. Thapaliya, P. Bangalore, J. F. Lofstead, K. Mohror, and A. Moody, "Managing I/O interference in a shared burst buffer system," in *45th International Conference on Parallel Processing, ICPP 2016, Philadelphia, PA, USA, August 16-19, 2016*, 2016, pp. 416–425.

[36] Thinkparq and BeeGFS, "Beegfs the leading parallel cluster file system," 2018, https://www.beegfs.io/docs/BeeGFS_Flyer.pdf.

[37] M.-A. Vef, V. Tarasov, D. Hildebrand, and A. Brinkmann, "Challenges and solutions for tracing storage systems: A case study with spectrum scale," *ACM Trans. Storage*, vol. 14, no. 2, pp. 18:1–18:24, 2018.

[38] M. Vilayannur, P. Nath, and A. Sivasubramaniam, "Providing tunable consistency for a parallel file store," in *Proceedings of the FAST '05 Conference on File and Storage Technologies, December 13-16, 2005, San Francisco, California, USA*, 2005.

[39] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. Miller, D. Long, and T. McLarty, "File system workload analysis for large scale scientific computing applications," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2004.

[40] T. Wang, K. Mohror, A. Moody, K. Sato, and W. Yu, "An ephemeral burst-buffer file system for scientific applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016*, 2016, pp. 807–818.

[41] J. Xing, J. Xiong, N. Sun, and J. Ma, "Adaptive and scalable metadata management to support a trillion files," in *Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2009, November 14-20, 2009, Portland, Oregon, USA*, 2009.

[42] S. Yang, W. B. Ligon III, and E. C. Quarles, "Scalable distributed directory implementation on orange file system," *Proc. IEEE Intl. Wrkshp. Storage Network Architecture and Parallel I/Os (SNAPI)*, 2011.