USER-LEVEL FILE SYSTEMS DEPLOYMENT IN HPC BETWEEN IMPROVEMENTS AND LIMITATIONS

Njoud O. Almaaitah
Computer Science Department, Information Technology Faculty, Mutah University
Al-Karak, Jordan
njoudmaitah@mutah.edu.jo

ABSTRACT— *Data-driven applications, in High-performance computing systems, involve a significant number of metadata operations, irregular access patterns, and small I/O requests. As a result, traditional parallel file systems became unable to efficiently handle such recent workloads, and data-driven applications suffered from significant I/O latency, lower throughput, and longer wait times. User-level file systems enhance the overall performance of High-performance computing clusters since the deployment overhead of such file systems is low compared with the application runtime. However, the use of a user-level file system is not without cost when used as part of a job or workflow. This is because it provides a new environment without data once it is deployed. Therefore, the input data must be copied from the parallel file system to the user-level file system before the computing job can start, and the output data must be copied in reverse from the user-level file system back to the parallel file system once the computing job is finished. These operations are referred to as stage-in and stage-out. In addition, a set of underlying technological difficulties are introduced either by High-performance computing platforms or application workloads. This paper introduces the most important limitations that could face user-level file systems when deployed in High-performance computing clusters.*

Keywords— HPC, PFS, user-level file systems**.**

I. INTRODUCTION

High-Performance Computing (HPC) term refers to all the facts of technology, methodology, and applications associated with achieving the highest computing power at any given time and technology. It aims to run complicated computational problems (i.e., workloads) as fast as possible on supercomputers. This action is commonly referred to as "supercomputing" [1]. Figure 1 shows an example of an HPC cluster, tiered as separate sets of computing and storage nodes, offering massive CPU cores, low-latency interconnects, and fast concurrent data bandwidth, typically managed by an underlying parallel file system. In some cases, a single misbehaving application can slow down the entire HPC system by interfering with other applications that use the same I/O subsystem. Such a situation is likely to become more common with larger and more powerful HPC systems [2].

A file system is the part of the operating system responsible for managing files and the resources on which these are stored. Effective computing would be near impossible without file systems. It allows efficient use of back-end storage devices, provides file sharing across various users and programs, and provides an abstract view of files and directories [3].

Parallel I/O systems appeared in the 1990s with two types of proposals: I/O libraries and parallel file systems (PFSs). The libraries which provide parallel access to data were proposed in NetCFD [4] and MPI-IO [5]. On another side, a set of early parallel file systems were developed such as PPFS [6], Lustre [7], Vesta [8], PARFISYS [9], BurstFS [10], Galley [11], BeeGFS [12]. HPC systems use such PFSs as storage subsystems where the data is shared by all users in a parallel manner [2]. However, providing a robust and effective file system is one of the critical issues in HPC today, as a response to a wide range of scientific applications that produce and manipulate massive amounts of data [13].

While workloads, in the last few years, essentially performed sequential I/O operations on large files, the current data-driven applications involve a significant number of metadata operations, irregular access patterns, and small I/O requests. As a result, the mentioned examples of traditional parallel file systems became unable to efficiently handle such recent workloads, and data-driven applications suffered from significant I/O latency, lower throughput, and longer wait times [14].

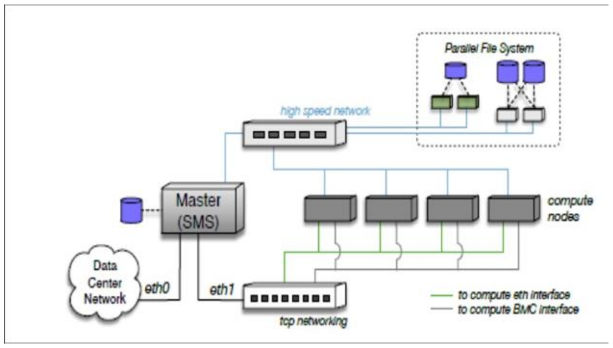

Figure 1. Overview of HPC cluster components.

Today, many supercomputers include SSDs, which can be utilized as dedicated burst buffers or as node-local burst buffers. The emergence of non-volatile storage media (e.g., NVRAM and SSDs) offers a great opportunity in such a setting. It is widely believed that NVRAM/SSDs will serve to bridge the I/O execution bottleneck between applications and hard disks because of their higher throughput and lower latency compared to hard disks [15].

Burst buffers are storage resources located between the parallel file system and the compute nodes. This new tier within the storage hierarchy closes the performance gap between node-local storage and parallel file systems. To exploit the bandwidth of burst buffers sufficiently, a number of file systems have been designed [16]. Furthermore, many user-level file systems emerged to exploit the fast storage resources (e.g. NVRAMs or SSDs) on nodes themselves.

User-level file systems can be implemented at the node level for particular HPC workloads to achieve high performance, unlike system-level file systems that are not optimized for HPC workloads and must be implemented by system administrators [17]. In this context, it is worth distinguishing between PFS, which has dedicated nodes and is considered a global file system shared by the users of the HPC system, and the user-level file systems (i.e., on-demand file systems) that use local storage of the nodes and is created for a specific application and deleted when the application is finished [2]. Figure 2 shows the improvement of the memory hierarchy of HPC systems that recently involves nodes SSDs and NVMe

This paper gives an overview of user-level file systems and their role in improving HPC systems by discussing two of them DeltaFS and MarFS. It also identifies most of the constraints that could limit the use and performance of user-level file systems. The rest of this paper is organized as follows: Section 2 contains the literature review; Section 3 explains the work of recent user-level file systems. Section 4 explains the limitations that may affect a user-level file system when it is deployed on HPC systems.

II. LITERATURE REVIEW

Parallel file systems have been developed, as a result of the necessity for high-throughput and concurrent read/write capabilities in HPC applications. One of the most common examples is the Lustre parallel file system [18] which is developed to provide I/O performance and overcome the restrictions of classical storage technologies. While all clients in Lustre architecture do the same functionalities, the servers play a variety of roles; for example, metadata servers handle the file system's metadata, and the object storage servers that maintain the file system's data in the form of objects [19]. This means that Lustre has separate metadata and object storage and implements a client-server approach with a server-side and distributed lock controller that ensures client coherency [7]. While Lustre is completely POSIX compatible, all transactions are handled atomically. This

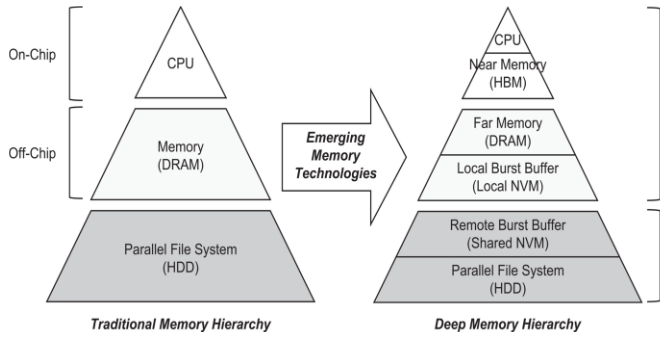
On-Chip
Off-Chip
CPU
Memory (DRAM)
Parallel File System (HDD)
Traditional Memory Hierarchy
Emerging Memory Technologies
CPU
Near Memory (HBM)
Far Memory (DRAM)
Local Burst Buffer (Local NVM)
Remote Burst Buffer (Shared NVM)
Parallel File System (HDD)
Deep Memory Hierarchy

Figure 2. Memory hierarchy with node-local storage (SSD, NVme, and DRAM)

means that all I/O requests are processed in order, without interruption, to avoid conflicts. On the other hand, it employs a distributed lock manager to handle the thousands of authorized clients attempting to access the file system in order to achieve parallelism. To be more specific, each component of the Lustre file system includes an instance of the Lustre distributed lock manager [20]. As a result of its excellence, Lustre has been deployed on more than half of the world's fastest supercomputers [7].

Another example of PFS is BeeGFS which began in 2004as a project to integrate Lustre PFS in a video streaming context and as storage in a 64-node Linux cluster with the objective of offering a scalable multi-threaded architecture [21].

BeeGFS provides some kind of flexibility, the client is constructed as a LINUX kernel module with additional software components that had been added to user space [22]. The architecture of BeeGFS includes the client side which is implemented with its private caching method that is staying up to date and tracking kernel changes. Besides clients, there are the metadata server, storage server, and management server[21, 23].

BurstFS, a distributed burst buffer file system, is proposed in [24] as a way to deliver high and scalable performance for burst I/O demands in scientific applications. BurstFS increased bandwidths by 1.83 and had strong scalability, according to preliminary testing.

One of the main shortages in implementing PFS in the HPC environment is obtaining a sufficient level of reliability and data integrity. Others investigated this issue and introduced a general tool for analyzing the failure handling of PFS named as PFault [25].

PFS is dependent heavily on POSIX I/O API since it is common between various Unix flavors. Because POSIX I/O API was the only option for a long time, most parallel applications and libraries (i.e. MPI-IO) were written to be compatible with POSIX API. Indeed, POSIX API was designed, from the start, for sequential applications and has many features that limit the performance of parallel applications [26]. However, many studies have proposed ways to use node-local storage devices such as SSD or NVMe to set up user-level, on-demand, and private parallel file systems for an HPC application. User-level file systems are designed to provide the efficiency and throughput required for concurrent data-driven applications [2].

III. USER-LEVEL FILE SYSTEMS

Since implementing traditional PFS on the HPC systems limits the performance, many researchers proposed solutions to create user-level, on-demand, and private parallel file systems for an HPC workload by utilizing node-local storage devices such as SSD or NVMe. User-level file systems are designed to achieve the required throughput and performance for parallel data-driven applications [2].

In contrast to the fact that POSIX file system semantics are used by the majority of large data centers for legacy workloads. Los Alamos National Laboratory's MarFS open -source initiative provides cloud-style object storage as a scalable POSIX-like file system. MarFS [27] was created to offer a cost-effective alternative to the expensive bandwidth of tape and the expensive capacity of PFS. It can store data sets for years at rates of up to 100 GB/sec. By distributing data among a large number of objects, or even multiple object systems, MarFS expands data capacity and bandwidth. It is developed to scale capacity and bandwidth in two dimensions for metadata.

DeltaFS [28] is another example of a user-level file system that avoids the need for dedicated servers to host data or metadata. Alternatively, the client middleware of a parallel job creates a file system namespace service that uses only scalable object storage and communicates with other jobs by sharing or distributing namespace snapshots. This server-less file system design is proven through experiments that it performs metadata operations of magnitude faster than conventional parallel file systems.

IV. LIMITATIONS

The use of user-level file systems in HPC clusters improves overall performance because the overhead of such file systems is low compared to the application runtime. In spite of that, user-level storage systems deployment is not without cost when used as part of a job or workflow. This is because such a storage system provides a new environment without data as soon as it is initiated. Such limitations that have faced the development and deployment of user-level file systems are summarized as follows:

A. Compatibility of storage devices

Storage devices are one of the critical resources that user-level file systems must handle. Local file systems like EXT4 and XFS have long been used as an abstraction point between distributed service daemons and local storage resources in parallel and distributed file systems. Local file system abstractions are still a viable and highly portable design option, but the development of new storage device technology has sparked a fresh round of research into alternate interfaces [29].

Persistent memory devices, for example, have more in common with dynamic memory than spinning magnetic media, and therefore may be accessed more effectively through direct user-space load/store operations rather than through indirect kernel-space block device and page cache activities. This feature has led to the development of storage abstractions like the Persistent Memory Developers Kit (PMDK), which provide simple transactional storage primitives on top of memory-mapped devices rather than block devices [30].

Lower-latency access routes can also benefit from faster block device interfaces. For example, the Storage Performance Development Kit (SPDK) offers an alternative interface for NVMe devices that uses user-space device drivers to reduce latency. PMDK and SPDK are designed for two different storage systems, but they both have the same goal: to reduce access costs for devices that don’t follow the design assumptions and performance characteristics of traditional hard drives [31].

B. HPC performance at different network protocols

The network is an essential part of any HPC cluster. Therefore, network performance evaluation at various protocol levels is critical for the management and administration of such clusters since it allows the networking system to be validated [32].

In regard to the hardware side, HPC applications (e.g. science and engineering simulation codes) nature needs to be operated through node-to-node communication and require message passing during execution. A new high interconnects for HPC was introduced in which relay on Omni-path which is considered as a compatible software with Open Fabrics Alliance stack for RDMA fabrics [33].

Over the years, the widest network abstraction model was TCP/IP sockets. Unfortunately, TCP/IP sockets lack to required features to adapt to the newest generation of HPC systems which are supported by powerful multi-core and many-core processing elements in order to achieve, the most common metrics, low latency, and high bandwidth [34, 33]. On another side, MPI is not easily suited to system services since it was built for application-level usage. OFI/libfabric, UCX, and Portals are more generic HPC network fabric abstractions that are not linked to MPI semantics or programming paradigms, making them better fundamental building blocks for ad-hoc file system implementations [35, 36]. Remote procedure calls and Mercury frameworks can be used with network abstractions to make creating ad-hoc file system services easier. gRPC and Apache Thrift are examples of general-purpose distributed computing [37, 38].

C. Resource consumption minimization

The concept of a user-level file system refers to a file system that is provided on-demand within an existing system rather than having dedicated resources. As a result, such a file system
must be able to coexist with other services or even application processes without consuming too much memory. The most common resource consumption issues such as busy polling of network resources; memory consumption; CPU or NUMA contention, should be addressed to get the required high performance [39].

V. CONCLUSION

User-level, on-demand, private parallel file systems for HPC workloads leverage node-local storage devices such as SSD or NVme. User-level file systems are designed to improve throughput and performance for parallel data-intensive applications. Two user-level solutions have been discussed inthis research: DeltaFs [28] and MarFS [27], but their deployment and execution of them, in complex HPC environments, are still limited by: the nature of network protocols, storage device compatibility, and high consumption of node-local resources.

REFERENCES

[1] T. Sterling, M. Brodowicz, and M. Anderson, High-performance computing: modern systems and practices. Morgan Kaufmann, **2017**.

[2] M. Soysal, M. Berghoff, T. Zirwes, M.-A. Vef, S. Oeste,

[3] A. Brinkmann, W. E. Nagel, and A. Streit, "Using on-demand file

[4] systems in hpc environments," in **2019** *International Conference on*

[5] *High Performance Computing & Simulation (HPCS).* IEEE, **2019**,

[6] pp. 390–398.

[7] L. F. Bic and A. C. Shaw, Operating systems principles. *Prentice-*

[8] *Hall, Inc.*, **2002**.

[9] R. Rew and G. Davis, “Netcdf: an interface for scientific data access,”

[10] *IEEE computer graphics and applications*, **vol. 10, no. 4,** pp. 76–82,

[11] **1990**.

[12] D. Hildebrand, A. Nisar, and R. Haskin, “pnfs, posix, and mpi-io: a
[13] tale of three semantics," in *Proceedings of the 4th Annual Workshop*
[14] *on Petascale Data Storage*, **2009**, pp. 32–36.

[15] J. V. Huber Jr, A. A. Chien, C. L. Elford, D. S. Blumenthal, and D. A.
[16] Reed, "Ppfs: A high performance portable parallel file system," in
[17] *Proceedings of the 9th International Conference on Supercomputing*,
[18] **1995**, pp. 385–394.
[19] T. Zhao, V. March, S. Dong, and S. See, “Evaluation of a performance
[20] model of lustre file system," *in 2010 Fifth Annual ChinaGrid Confer-*
[21] *ence. IEE*E, **2010**, pp. 191–196.
[22] P. F. Corbett, S. J. Baylor, and D. G. Feitelson, “Overview of the vesta
[23] parallel file system*," ACM SIGARCH Computer Architecture News*,
[24] vol. 21, no. 5, pp. 7–14, **1993**.
[25] F. Pérez, J. Carretero, F. García, P. De Miguel, and L. Alonso, “Eval-
[26] uating parfisys: A high-performance parallel and distributed file sys-
[27] tem," *Journal of systems architecture*, vol. 43, no. 8, pp. 533–542,
[28] **1997**.
[29] T. Wang, W. Yu, K. Sato, A. Moody, and K. Mohror, “Burstfs: A dis-
[30] tributed burst buffer file system for scientific applications," Lawrence
[31] Livermore National Lab.(LLNL), *Livermore*, CA (United States),
[32] Tech. Rep., **2016**.
[33] N. Nieuwejaar and D. Kotz, “Performance of the gallery parallel file
[34] system," in Proceedings of the fourth workshop on I/O in parallel and
[35] distributed systems: *part of the federated computing research confer-*
[36] *ence*, **1996**, pp. 83–94.
[37] J. Heichler, “An introduction to beegfs,” **2014**.
[38] J. Lüttgau, M. Kuhn, K. Duwe, Y. Alforov, E. Betke, J. Kunkel, and
[39] T. Ludwig, "Survey of storage systems for high-performance comput-
[40] ing," *Supercomputing Frontiers and Innovations*, vol. 5, no. 1, **2018**.
[41] M.-A. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes,
[42] and A. Brinkmann, "GekkoFS-a temporary distributed file system for
[43] hpc applications," in 2018 *IEEE International Conference on Cluster*
[44] *Computing (CLUSTER)*. IEEE, **2018**, pp. 319–324.
[45] Y. Kim, A. Gupta, and B. Urgaonkar, “Mixedstore: An enterprise-
[46] scale storage system combining solid-state and hard disk drives,"
[47] Citeer, **2008**.
[48] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. R. De Supin-
[49] ski, N. Maruyama, and S. Matsuoka, "A user-level infiniband-based
[50] file system and checkpoint strategy for burst buffers," in 2014 14th
[51] IEEE/ACM International Symposium on Cluster, Cloud and Grid
[52] Computing. IEEE, **2014**, pp. 21–30.
[53] A. Brinkmann, K. Mohror, and W. Yu, “Challenges and Opportunities
[54] of User-Level File Systemsfor HPC," *Lawrence Livermore National*
[55] *Lab.(LLNL), Livermore, CA (United States)*, Tech. Rep., **2017**.
[56] P. Schwan et al., “Lustre: Building a file system for 1000-node clus-
[57] ters," *in Proceedings of the **2003** Linux symposium*, vol. **2003**, 2003,
[58] pp. 380–386.
[59] P. Braam, “The lustre storage architecture,” arXiv preprint
[60] arXiv:1903.01955, **2019**.
[61] T. K. Petersen, “Inside the lustre file system,” SEAGATE Technology
[62] paper, **2015**.
[63] F. Herold, S. Breuner, and J. Heichler, “An introduction to BeeGFS,”**2014**.
[64] M. Cintra, “Seventh framework programme,” Assessment, 2016.
[65] “Home,” Mar **2021**. [Online]. Available: https://www.beegfs.io/c/
[66] T. Wang, K. Mohror, A. Moody, W. Yu, and K. Sato, “BurstFS: A
[67] distributed burst buffer file system for scientific applications," in The
[68] *International Conference for High Performance Computing*, Networking, Storage and Analysis (SC poster), **2015**.
[69] J. Cao, O. R. Gatla, M. Zheng, D. Dai, V. Eswarappa, Y. Mu, and
[70] Y. Chen, "Pfault: A general framework for analyzing the reliability
[71] of high-performance parallel file systems*," in Proceedings of the 2018*
[72] *International Conference on Supercomputing*, **2018**, pp. 1–11.
[73] R. Ross, R. Thakur, and A. Choudhary, “Achievements and challenges
[74] for i/o in computational science," *in Journal of Physics: Conference*
[75] *Series. IOP Publishing*, **2005**, p. 069.
[76] J. T. Inman, W. F. Vining, G. W. Ransom, and G. A. Grider, “Marfs,
[77] a near-posix interface to cloud objects," ; Login, vol. 42, no. LA-UR-
[78] 16-28720; LA-UR-16-28952, **2017**.
[79] Q. Zheng, K. Ren, G. Gibson, B. W. Settlemyer, and G. Grider,
[80] "Deltafs: Exascale file systems scale better without dedicated servers,"
[81] *in Proceedings of the 10th Parallel Data Storage Workshop*, **2015**, pp.
[82] 1–6.
[83] L. Benedicic, F. A. Cruz, A. Madonna, and K. Mariotti, “Sarus: Highly
[84] scalable docker containers for hpc systems," *in International Confer-*

[85] *ence on High Performance Computing*. Springer, **2019**, pp. 46–60.

[86] H.-b. Chen, Z. Qiao, and S. Fu, “Applying SDN based data network

[87] on HPC Big Data Computing–Design, Implementation, and Evalua-

[88] tion," in **2019** *IEEE International Conference on Big Data* (Big Data).

[89] IEEE, **2019**, pp. 6007–6009.

[90] U. Yang, P. Luszczek, S. Baley, and K. Teranishi, “An introduction to

[91] the xSDK a community of diverse numerical HPC software packages."

[92] *Sandia National Lab.(SNL-CA), Livermore*, CA (United States), Tech. Rep., **2019**.

[93] E. Gamess and H. Ortiz-Zuazaga, “Evaluation of point-to-point net-

[94] work performance of hpc clusters at the level of udp, tcp, and mpi,"

[95] *in IV Simposio Científico y Tecnológico en Computación, Caracas,*

[96] *Venezuela*, **2016**.

[97] M. Feldman and A. Snell, “A new high performance fabric for hpc,”

[98] *Intersect360 Research white paper*, **2016**.

[99] G. Maglione-Mathey, P. Yebenes, J. Escudero-Sahuquillo, P. J. Gar-

[100] cia, F. J. Quiles, and E. Zahavi, "Scalable deadlock-free determinis-

[101] tic minimal-path routing engine for infiniband-based dragonfly net-

[102] works," *IEEE Transactions on Parallel and Distributed Systems*,

[103] vol. 29, no. 1, pp. 183–197, **2017**.

[104] R. Latham, R. Ross, and R. Thakur, “Can MPI be used for persis-

[105] tent parallel services?" in European Parallel Virtual Machine/Message

[106] Passing Interface Users' *Group Meeting. Springer*, **2006**, pp. 275–

[107] 284.

[108] P. Grun, S. Hefty, S. Sur, D. Goodell, R. D. Russell, H. Pritchard, and

[109] J. M. Squyres, "A brief introduction to the openfabrics interfaces-a

[110] new network api for maximizing high performance application effi-

[111] ciency*," in 2015 IEEE 23rd Annual Symposium on High-Performance*

[112] *Interconnects. IEEE*, **2015**, pp. 34–39.

[113] B. W. Barrett, R. Brightwell, S. Hemmert, K. Pedretti, K. Wheeler,

[114] K. D. Underwood, R. Reisen, A. B. Maccabe, and T. Hudson, "The

[115] Portals 4.0 network programming interface," *Sandia National Labora-*

[116] *tories*, **2019**.

[117] J. Soumagne, D. Kimpe, J. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. Ross, “Mercury: Enabling remote procedure call for high performance computing,” in **2013** *IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, **2013**, pp. 1–8.

[118] A. Brinkmann, K. Mohror, W. Yu, P. Carns, T. Cortes, S. A. Klasky,

[119] A. Miranda, F.-J. Pfreundt, R. B. Ross, and M.-A. Vef, "Ad hoc file

[120] *systems for high-performance computing," Journal of Computer Sci-*

[121] *ence and Technology*, vol. 35, no. 1, pp. 4–26, **2020**.