Network-accelerated Distributed File Systems

Salvatore Di Girolamo, Daniele De Sensi, Konstantin Taranov, Milos Malesevic, Maciej Besta,
Timo Schneider, Severin Kistler, Torsten Hoefler
firstname.lastname@ethz.com
ETH Zurich, Switzerland

ACM Reference Format:
Salvatore Di Girolamo, Daniele De Sensi, Konstantin Taranov, Milos Malesevic, Maciej Besta,, Timo Schneider, Severin Kistler, Torsten Hoefler. 2021. Network-accelerated Distributed File Systems. In *Proceedings of PDSW '21: 6th International Parallel Data Systems Workshop (PDSW '21).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 EXTENDED ABSTRACT

Fast, scalable, and reliable storage is a first-class requirement of both HPC systems and datacenters. Applications constantly interface the file system to read input data or write newly produced one. This happens not only in the startup or finalization phases but throughout the application's lifespan. Examples of applications heavily interfacing the file system are deep neural network training and scientific simulations. In the former, I/O overheads can account for up to 60% of the total running time [8] (e.g., reading training datasets) while in the latter it can reach up to 90% of the runtime [5] (e.g., writing and reading checkpoints, writing simulation results).

Distribute File Systems (DFSs) play a fundamental role in tackling the I/O bottleneck. By decoupling control and data plane, these architectures can be easily managed and scaled out. Storage policies are defined in the control plane, and express how data must be stored (e.g., replicated or erasure-coded) and how it can be accessed (e.g., client authentication, permissions). For example, if and how a file must be replicated or erasure-coded is defined in the control plane, while the actual replication or erasure coding (EC) process happens in the data plane.

Until recently, the performance of data-path storage operations has been greatly limited by the performance of the storage media (e.g., disks or SSDs). This led to the introduction of complex software layers in DFSs for avoiding this bottleneck with optimizations such as batching and striping, and efficiently enforcing storage policies. As accesses were dominated by storage performance, other factors like network performance, multiple data copies, and CPU utilization became of negligible importance. However, with the emergence of dense, byte-addressable non-volatile main memories (NVMMs) [10], this basic assumption must be revised. In fact, NVMMs have performance characteristics close to DRAM [6], and are several orders of magnitude faster than hard drives and SSDs. Suddenly, factors as network performance and software overheads play again an important role and must be optimized to not let them become bottlenecks.

For this reason, remote direct memory access (RDMA) [1] has been the focus of many DFS optimizations [2, 3, 9, 11, 12]. RDMA is designed to provide low latency and high bandwidth one-sided communications, where nodes can access memory of remote peers (in the DFS case, of the storage nodes) without involving their CPUs. Since the use of high-level libraries that utilize RDMA but hide RDMA accesses produce limited benefits to DFSs, researchers started proposing DFSs that have low-level RDMA operations in their core design [7, 13, 14].

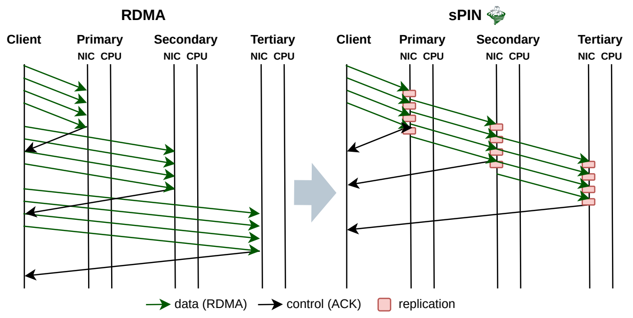

Figure 1: Example of a DFS policy (data replication). In the RDMA case (left), the client writes to each replica node. With sPIN (right), the policy is offloaded to the NIC of the storage nodes, that propagate the data on a per-packet basis.

Even though RDMA communications do not involve the CPU of the storage node, DFS policies cannot be enforced unless the CPU of the storage node is notified through, e.g., remote procedure calls (RPC), possibly losing the zero-copy benefits of RDMA. Alternatively, some DFS policies such as replication can be enforced at the client, even though this might limit their efficiency. For example, Figure 1 (left) shows the case where the client enforces replication by sending data to all storage nodes acting as replicas.

We investigate how post-RDMA solutions like sPIN, which enable applications to install custom packet handlers on the network interface card (NIC), can be used to run DFS policies directly on the data path. In this way, we keep the benefits of RDMA (no CPU involvement, low latency, and high throughput), while being able to apply DFS policies on a per-packet basis. Figure 1 (right) shows an overview of our approach for the data replication policy: instead of forcing the client to send the data to each replica, data is forwarded directly by the NICs of the storage nodes, on a per-packet basis. There, sPIN packet handlers run DFS policies, eventually forwarding the packets to othre storage nodes in case of data replication or erasure coding is needed. No per-client hard state is kept on the NIC, avoiding additional network round trips for, e.g., establishing connections. By using PsPIN [4], an open-source implementation of sPIN, we show how processing packets on the data path can produce latency improvements for writes (up to 2x), data replication (up to 4x), and erasure coding (up to 3x), when compared to comparable non-offloaded versions.

PDSW '21, November 15, 2021, St. Louis, MO
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

REFERENCES

[1] InfiniBand Trade Association. 2004. InfiniBand Architecture Specification, Volume 1, Release 1.2.

[2] Dhruba Borthakur et al. 2008. HDFS architecture guide. *Hadoop Apache Project* 53, 1-13 (2008), 2.

[3] Peter Braam. 2019. The Lustre storage architecture. *arXiv preprint arXiv:1903.01955* (2019).

[4] Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beranek, Luca Benini, and Torsten Hoefler. 2021. A RISC-V in-network accelerator for flexible high-performance low-power packet processing. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*.

[5] Sriram Lakshminarasimhan, David A Boyuka, Saurabh V Pendse, Xiaocheng Zou, John Jenkins, Venkatram Vishwanath, Michael E Papka, and Nagiza F Samatova. 2013. Scalable in situ scientific data encoding for analytical query processing. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*. 1–12.

[6] Hai-Kun Liu, Di Chen, Hai Jin, Xiao-Fei Liao, Binsheng He, Kan Hu, and Yu Zhang. 2021. A Survey of Non-Volatile Main Memory Technologies: State-of-the-Arts, Practices, and Future Directions. *Journal of Computer Science and Technology* 36, 1 (2021), 4–32.

[7] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. 2017. Octopus: an RDMA-enabled distributed persistent memory file system. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 773–785.

[8] Sarunya Pumma, Min Si, Wu-chun Feng, and Pavan Balaji. 2017. Parallel I/O optimizations for scalable deep learning. In *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 720–729.

[9] Robert B Ross, Rajeev Thakur, et al. 2000. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th annual Linux showcase and conference*. 391–430.

[10] Arthur Sainio. 2016. NVDIMM: changes are here so what's next. *Memory Computing Summit* (2016).

[11] Frank B Schmuck and Roger L Haskin. 2002. GPFS: A Shared-Disk File System for Large Computing Clusters.. In *FAST*, Vol. 2.

[12] Lizhe Wang, Yan Ma, Albert Y Zomaya, Rajiv Ranjan, and Dan Chen. 2014. A parallel file system with application-aware data layout policies for massive remote sensing image processing in digital earth. *IEEE Transactions on Parallel and Distributed Systems* 26, 6 (2014), 1497–1508.

[13] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2019. Orion: A distributed file system for non-volatile main memory and RDMA-capable networks. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. 221–234.

[14] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2020. FileMR: Rethinking RDMA Networking for Scalable Persistent Memory. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 111–125.