Zebra: An Efficient, RDMA-Enabled Distributed Persistent Memory File System

Jingyu Wang[1], Shengan Zheng[2(✉)], Ziyi Lin[3], Yuting Chen[1], and Linpeng Huang[1(✉)]

[1] Shanghai Jiao Tong University, Shanghai, China
{wjy114,chenyt,lphuang}@sjtu.edu.cn
[2] MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University, Shanghai, China
shengan@sjtu.edu.cn
[3] Alibaba Group, Hangzhou, China
cengfeng.lzy@alibaba-inc.com

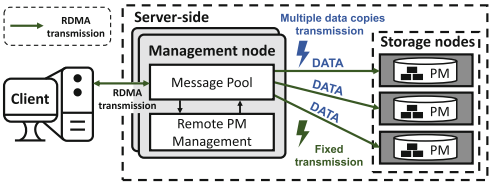
Abstract. Distributed file systems (DFSs) play important roles in datacenters. Recent advances in persistent memory (PM) and remote direct memory access (RDMA) technologies provide opportunities in enhancing distributed file systems. However, state-of-the-art distributed PM file systems (DPMFSs) still suffer from *a duplication problem* and *a fixed transmission problem*, leading to high network latency and low transmission throughput. To tackle these two problems, we propose *Zebra*, an efficient RDMA-enabled distributed PM file system—Zebra uses *a replication group design* for alleviating the heavy replication overhead, and leverages *a novel transmission protocol* for adaptively transmitting file replications among nodes, eliminating the fixed transmission problem. We implement Zebra and evaluate its performance against state-of-the-art distributed file systems on an Intel Optane DC PM platform. The evaluation results show that Zebra outperforms CephFS, GlusterFS, and NFS by 4.38×, 5.61×, and 2.71× on average in throughput, respectively.

Keywords: RDMA · Adaptive transmission · Persistent memory · Distributed file system

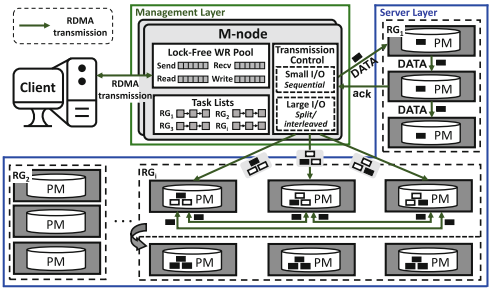
1 Introduction

Nowadays, distributed file system plays an increasingly important role in datacenters. A distributed file system is a client/server-based application that allows clients to access and process remote files as they do with the local ones. Many efforts have been spent on leveraging persistent memory (PM) and Remote Direct Memory Access (RDMA) technologies to enhance the distributed file systems. Persistent memories are of large capacities and near-DRAM performance,

A. Bhattacharya et al. (Eds.): DASFAA 2022, LNCS 13245, pp. 341–349, 2022.
https://doi.org/10.1007/978-3-031-00123-9_28



(a) **Architecture of a typical DPMFS.** It adopts a fixed transmission mode, suffering from heavy replication overhead between the management and the storage nodes.



(b) **Architecture of Zebra.** We let each replication group contain three D-nodes and each file has three file replications.

Fig. 1. A comparison of a typical DPMFS and Zebra.

while RDMA supports direct accesses of PMs. PM and RDMA supplement each other, enabling efficient access to remote files.

Several RDMA-enabled distributed PM file systems (DPMFS), such as Octopus [5], Orion [12] and Assise [2] achieve high performance through coupling PM and RDMA features. A DPMFS, as Fig. 1(a) shows, is composed of a management node and many storage nodes. The management node is responsible for receiving clients' requests and/or managing file replications. The storage nodes are organized into a chain structure or a star topology, on which file replications are stored. Persistent memories are deployed on the central and storage nodes, as they are of large capacities and near-DRAM accessing speeds, and RDMA is employed, allowing users to directly access PMs and efficiently synchronize file replications [6]. Conventional distributed file systems adopt RDMA by substituting their communication modules with RDMA libraries.

The problem of file replication can be cast into a problem of file transmission: file replications are transmitted from the management node to the storage nodes, or vice versa, such that clients can access files close to them. Meanwhile, the performance of existing DPMFSs suffers from two transmission problems.

Problem 1: Duplication Problem. File replications need to be intensively transmitted. In a chain-structured DPMFS [4,9], the management node is followed by

a chain of storage nodes—a file is transmitted along the chain and much effort needs to be spent on synchronizing file replications so as to avoid inconsistency. Comparatively, in a DPMFS with a star topology, file replications are only transmitted between the central node and the storage nodes [4,8,11], whilst the loads are not balanced, as the central node suffers from heavy replication overhead.

Problem 2: Fixed Transmission Problem. A DPMFS, or even a traditional DFS, usually adopts a fixed transmission strategy. Such a DPMFS does not adapt to different granularity, and thus suffers from a mismatch between the file replications and the transmission block sizes. A DPMFS needs to adopt a much more flexible transmission strategy in small/large file transmission scenarios (in which files of small/large sizes are transmitted), binding files to different transmission modes.

To tackle the duplication and the fixed transmission problems, we present *Zebra*, an efficient, RDMA-enabled distributed PM file system. The key idea is to (1) use *a replication group design* for alleviating the heavy replication overhead, and (2) design *an adaptive RDMA-based transmission protocol* that adaptively transmits file replications among nodes, solving the fixed transmission problem. In addition, to support multithreaded data transmission, Zebra leverages a lock-free work request (WR) pool and a conflict resolution mechanism to accelerate transmission. This paper makes the following contributions:

- *Design:* Zebra uses a replication group design that alleviates the heavy replication overhead between the central node and storage nodes.
- *Protocol:* Zebra provides an adaptive RDMA transmission protocol for distributed PM file systems, significantly improving the transmission throughput.
- *Implementation and evaluation:* We implement Zebra and evaluate its performance against state-of-the-art distributed file systems on an Intel Optane DC PM platform. The evaluation results clearly show the efficiency of Zebra in small/large file transmission scenarios.

2 The Zebra System

2.1 Design

Zebra is an efficient DPMFS that uses an adaptive transmission mode to speed up accesses to PMs. As Fig. 1(b) shows, Zebra consists of a management layer containing management nodes (*M-nodes*) which are responsible for coordinating resources, and a server layer containing a set of storage nodes (*D-nodes*):

Group of D-nodes. Zebra provides a replication group (RG) mechanism that divides D-nodes into groups. The D-nodes in a group work as backups for each other. Each RG contains a master D-node that coordinates the D-nodes, guarantees their orderliness, and feedback to the M-node.

M-nodes. Zebra decouples its metadata and data to improve the scalability and performance. File I/O requests are received, processed, and broadcasted by the M-node.

Zebra also pre-allocates memories for work requests (WR) during the registration phase, reducing the overhead of memory allocation during RDMA transmissions. As Fig. 2 shows, Zebra uses a *lock-free WR pool* for storing WRs, which is produced for each transfer.

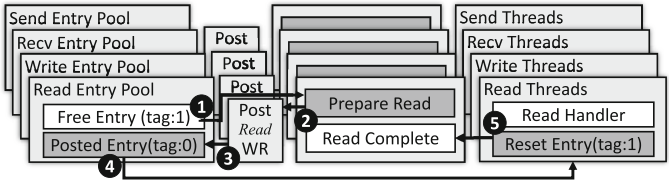


Fig. 2. Workflow in a lock-free work request pool.

2.2 An Adaptive Replication Transmission Protocol

All of the nodes in Zebra system are connected through an RDMA network. During file transmission, Zebra first establishes a *critical transmission process* that transmits data and transmission control information from the M-node to one or more D-nodes; Zebra then asynchronously transmits data among D-nodes, exchanging file replications.

Let $\{d_1, d_2, \dots\}$ be a set of data packets to be transmitted. Let $g(D_i)$ be the D-node D_i in a replication group g. Zebra flexibly chooses the transmission modes on the basis of file transmission scenarios, adapting to I/O sizes:

1) When transmitting a small number of files and/or files of small sizes, Zebra uses a *sequential transmission* mode due to the sufficiency of the transmission bandwidth. The file replications can be transmitted directly to the D-nodes using a chain replication style, where (1) the M-node M picks up a D-node, say D_i, as a master D-node and writes the data to it; (2) D_i receives the transmission control information and writes data to the other n nodes in the RG g. D_{i-2} (the D-node next to last) sends an ACK message to the M-node for completing the writing activity.

2) Zebra uses a novel, *split/interleaved transmission* mode in large I/O scenarios (when a large number of files and/or files of large sizes are transmitted). This mode reduces the transmission overhead from the M-node to the D-nodes. First, Zebra takes a split transmission process, $M \rightarrow g(D_1) = \{d_1, d_2, ..., d_k\}$, $M \rightarrow g(D_2) = \{d_{k+1}, d_{k+2}, ..., d_j\}, \dots$ in which the data is split and transmitted from the M-node M to the D-nodes. The successive interleaved transmission are $g(D_1) \leftrightarrow g(D_2) \leftrightarrow \cdots \leftrightarrow g(D_n)$, indicating that the file replications are interleaved between the D-nodes in the replication group, rather than between the M-node and the replication group.

For example, the upper bound of the RDMA throughput (T_R) is approximately 12 GB/s in Mellanox ConnectX-5, whereas the bandwidth of PM (T_{PM}) is approximately 35 GB/s in read and 20 GB/s in write. In the case, even though the

maximal bandwidth of RDMA is reached, the bandwidth of PM is not fully leveraged. The split/interleaved mode expands the transmission bandwidth, reducing the latency to $L_s = \frac{S_d/m}{T_R}$, where S_d is the total size of data in transmission and the data is split into m parts. Here we let the number of splits be $m = \frac{T_{PM}}{T_R}$, where T_{PM} is the maximal bandwidth of PM. The latency of transmitting unsplit data/files is m times of that of taking the *split/interleaved transmission* mode, i.e., $L_{us} = \frac{S_d}{T_R}$. The latency increases exponentially after the bandwidth reaches a bottleneck.

Load Balance. The M-node manages all space allocations in Zebra. It balances the storage usage and the throughput by sampling the realtime load across the RGs: the M-node gathers the operations; a load-balance controller records the total capacity and realtime usage of each RG, and computes the throughput of each node according to the number of ongoing RDMA operations.

Let g be a replication group and C_g be the percentage of its available capacity. Let $Task$ be the set of unfinished tasks and $Task_g$ be the set of unfinished tasks w.r.t. g. Let $availability$ of g be $availability_g = \alpha \times C_g + \beta \times \frac{|Task|}{|Task_g|} \times 100\%$, where α and β are two real values defined by human engineers. The RG with a smaller availability (i.e., with less running tasks) is assigned a higher possibility for storing file replications. An RG with the largest C is chosen when several RGs are of the same availability values. To avoid frequent calculations, Zebra selects RG candidates with the highest availability values and updates them periodically. This reduces the load-balancing overhead.

2.3 Multithreaded RDMA Transmission

Multithreaded transmission are frequent in DFSs. Traditionally, lock operations need to be performed in resource pools to guarantee the availability of resources, while it incurs excessive lock contention among multiple threads. Zebra takes a lock-free mechanism that uses kernel atomic verbs to manage WR resources in the WR pools. When Zebra performs a transmission, a daemon thread calls `atomic_dec_and_test` to search for an available WR entry.

In order to avoid conflicts caused by linear detection among multi-threading, we adopt a decentralized query strategy to speed up usable WR acquisition process. Threads process work requests scattered within the WR pool. Every time a thread obtains an available WR, it linearly searches and points to the next available WR in advance. When a thread detects that the current WR has been occupied, it searches for the next available WR. If a thread detects that the previously available WR is now occupied, and then performs a location hash. This ensures a high level of parallelism.

3 Evaluation

We evaluated the performance of Zebra against other state-of-the-art distributed file systems using a set of benchmarks. The evaluation is designed to answer the following research questions (Fig. 3):

– RQ1. How effective is Zebra in transmitting files of different sizes?
– RQ2. How effective is Zebra in processing multithreaded workloads, compared to the other distributed file systems?
– RQ3. Is Zebra scalable for real-world applications?

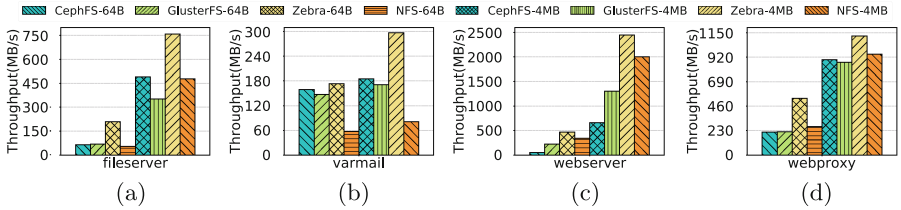


Fig. 3. Systems' throughput w.r.t. filebench with different workloads.

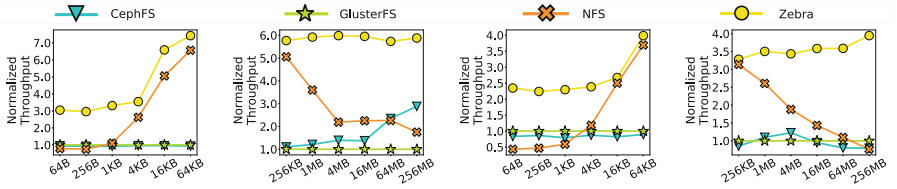
3.1 Setup

We conduct the evaluation on an Intel Optane DC PM platform. Each node is equipped with six 256 GB Intel Optane DCPMMs that are configured in the App-Direct interleaved mode. The nodes communicate with each other through a Mellanox ConnectX-5 RNIC with the Infiniband mode. The workload threads are pinned to local NUMA nodes. Zebra is deployed on a cluster, on which every two D-nodes are organized into a replication group.

We compare Zebra against four file systems: CephFS [11], GlusterFS [7], and NFS [3]. We use *Filebench* [10], to evaluate the adaptive transmission design in large/small file transmission scenarios. We use *fileserver*, a workload in *Filebench*, to emulate the multithreaded scenario. Furthermore, we evaluate the overall performance of Zebra using four *Filebench* workloads. In addition, we evaluate Zebra's scalability with Redis [1], which is a popular persistent key-value store.

3.2 Sensitivity to I/O Size

We use four filebench workloads (fileserver, varmail, webproxy, and webserver) to evaluate Zebra. For small I/O sizes (Fig. 4(a)(c) where the I/O size is smaller than 64 KB), Zebra adopts a sequential transmission mode. This reduces the overhead of one-to-many transmission, increasing the throughput. Figure 4(b)(d) corresponds to the scenario in which the I/O sizes are larger than 256 MB. The threshold of switching the transmission modes depends on the I/O size, which is also determined by the peak throughput of the other systems. This ensures that the use of efficient split/interleaved transmission can alleviate the performance degradation caused by throughput bottlenecks. In the case of large data transmission scenarios (i.e., 1 or 2 GB), the performance of CephFS, GlusterFS, and NFS decreases by 21%, 27% and 58% on average, respectively. The performance of Zebra decreases by only 2.3%.



(a) 1-thread S I/O (b) 1-thread L I/O (c) 16-thread S I/O (d) 16-thread L I/O

Fig. 4. Performance for various I/O sizes and threads. It is normalized w.r.t. the throughput of GlusterFS.

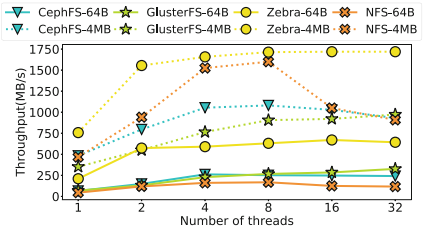


Fig. 5. Concurrency performance.

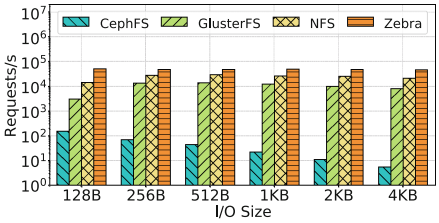


Fig. 6. Performance of running Redis.

In a small I/O size scenario, Zebra outperforms CephFS, GlusterFS, and NFS transmission protocols by 3.7×, 3.4×, and 2.9×, respectively. In a large I/O size scenario, Zebra outperforms CephFS, GlusterFS, and NFS by 4.7×, 6.2×, and 2.3×, respectively.

3.3 Concurrency

Figure 5 shows the performance of the systems running with multi threads. We set the I/O size to 64 B and 4 MB and run fileserver with the thread numbers ranging from 1 to 32. Zebra adopts a two-tier concurrency mechanism, including a lock-free WR pool and an adaptive transmission strategy that improves its scalability.

In CephFS, each replica applies the update locally and subsequently sends an acknowledgment to the primary node. This results in heavy network transmission on the critical path. GlusterFS performs the worst because of its heavy data management overhead. When NFS reaches the maximal number of threads, any subsequent requests need to wait, resulting in timeouts. As NFS is a TCP-based network file system, a large number of threads with unbalanced loads can lead to give up of file transmission. Zebra allows data to be transmitted in a multithreaded manner and achieves the highest performance among the file systems in the multithreaded transmission scenario. Zebra outperforms CephFS, GlusterFS, and NFS on average by 2.1×, 2.3×, and 1.8×, respectively.

3.4 Scalability

We use Redis [1], a popular persistent key-value store, to further evaluate the performance of Zebra in real world applications. We choose AOF mechanism in Redis to achieve data persistence, which synchronizes data after every modification. As shown in Fig. 6, Zebra achieved the optimum performance on the basis of optimization in small data transmission. The underlying storage consistency protocol of CephFS, makes it vulnerable to small data accesses. The performance of CephFS drops 98.2% from 64 B to 4096 B. Zebra and GlusterFS use block storage, which stores a set of data in chunks. Block storage provides high throughput against small data I/O and also guarantees preferable performance. Compared to the GlusterFS, Zebra gains a 17–21× throughput improvement.

4 Conclusion

This paper presents Zebra, an efficient RDMA-enabled DPMFS. The key idea of Zebra is to improve the efficiency of file replication by improving the efficiency of data transmission. Zebra leverages PMs for speeding up file accesses, uses an adaptive RDMA protocol for reducing network latency and improving transmission throughput. It uses the method of dividing large data into blocks and complete data reliability backup. The evaluation results show that Zebra outperforms other state-of-the-art systems, such as CephFS, GlusterFS, NFS, and Octopus in transmitting data streams and synchronizing file replications.

Acknowledgements. This research is supported by National Natural Science Foundation of China (Grant No. 62032004), Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX0102) and Natural Science Foundation of Shanghai (No. 21ZR1433600).

References

1. Redis (2018). https://redis.io/
2. Anderson, T.E., Canini, M., Kim, J., et al.: Assise: performance and availability via NVM colocation in a distributed file system. CoRR (2020)
3. Callaghan, B., Lingutla-Raj, T., Chiu, A., et al.: NFS over rdma. In: Network-I/O Convergence: Experience, Lessons, Implications (2003)
4. Davies, A., Orsaria, A.: Scale out with glusterfs. Linux J. (2013)
5. Lu, Y., Shu, J., Chen, Y., et al.: Octopus: an RDMA-enabled distributed persistent memory file system. In: ATC (2017)
6. Nielsen, L.H., Schlie, B., Lucani, D.E.: Towards an optimized cloud replication protocol. In: SmartCloud (2018)
7. Noronha, R., Panda, D.K.: IMCA: a high performance caching front-end for glusterfs on infiniband. In: ICPP (2008)
8. Shan, Y., Tsai, S.Y., Zhang, Y.: Distributed shared persistent memory. In: SOCC (2017)
9. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: MSST (2010)

10. Tarasov, V., Zadok, E., Shepler, S.: Filebench: a flexible framework for file system benchmarking. Usenix Mag. (2016)
11. Weil, S.A., Brandt, S.A., et al.: Ceph: a scalable, high-performance distributed file system. In: OSDI (2006)
12. Yang, J., Izraelevitz, J., Swanson, S.: Orion: a distributed file system for non-volatile main memory and RDMA-capable networks. In: FAST (2019)