IEICE TRANSACTIONS

on Information and Systems

DOI:10.1587/transinf.2024EDL8060

Publicized:2024/09/20

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

LETTER

Dalio: In-Kernel Centralized Replication for Key-Value Stores

Gyuyeong KIM[†a)], *Nonmember*

SUMMARY Replication is commonly used in distributed key-value stores for high availability. Recent works show that centralized replication provides high throughput through low-overhead write coordination and consistency-aware read forwarding. Unfortunately, they rely on specialized hardware, which is deploy-challenging and poses various limitations. To this end, we present Dalio, a software-based centralized replication system that does not require extra hardware while supporting high throughput. Our key idea is to offload the replication function to per-shard load balancers with eBPF, an emerging kernel-native technique. By building a replication coordinator with eBPF, we can avoid burdensome kernel networking stack overhead. Our experimental results show that Dalio achieves throughput better than the vanilla Linux by up to 2.05× and is comparable to a hardware-based solution.

key words: *Networking stacks, Replication protocol, In-kernel acceleration*

1. Introduction

Key-value stores are essential building blocks of modern online services. To mask failures, items in a data shard are often replicated over multiple servers, typically 3 replicas [1], [2]. Traditionally, replication is performed by distributed protocols. Leader-based protocols [3], [4] can achieve linearizability easily but suffer from poor performance as only the leader handles requests. To improve performance, leaderless replication [2], [5] allows any replica to handle reads and writes but fails to achieve linearly-increasing performance due to extra coordination overhead for linearizability.

A recent work [6] shows that performing centralized leaderless replication provides high performance and linearizability. In particular, it moves the entire replication function into the Top-of-Rack (ToR) switch. The switch coordinates writes among replicas by maintaining the list of temporarily inconsistent keys (i.e., dirty set) due to pending writes. The switch also tracks the number of aggregated write replies, and commits the write when all replies are aggregated. For reads, the switch sends requests to a random replica by default. However, if the requested key is included in the dirty set, the switch forwards the requests to the latest known consistent replica, which is also tracked in the switch. Write coordination overhead is reduced as the replication function is offloaded from storage servers to the network. In addition, linearizability can be ensured since reads can always be forwarded to a consistent replica, even if the key is inconsistent.

Unfortunately, the existing work has limitations as follows. First, they rely on specialized hardware like an Intel Tofino ASIC [7], which is not widely available in modern data centers. Second, they require all replicas to be under the same rack due to L2 multicast, which means the request cannot be delivered to other racks. Third, the resource constraints of switch hardware pose several limits on applications. For example, item key size is limited to 4 bytes, which is insufficient for typical workloads where key sizes are tens of bytes [8].

To this end, we present Dalio, a software-based centralized replication system that provides high performance without relying on specialized hardware. Our key idea is to build a replication coordinator in per-shard load balancers by leveraging the power of extended Berkeley Packet Filter (eBPF) [9]. The insights behind the idea are as follows. First, storage servers in the same replication group are connected via a software per-shard load balancer (PSLB) [10] that determines destination replicas. Therefore, PSLBs are a vantage point for centralized replication similar to the ToR switch.

Second, the emerging eBPF [9] allows us to build a high-performance replication coordinator in software. We can manipulate the packet header and payload in the host network stack by attaching a custom eBPF program to various hook points like the eXpress Data Path (XDP) layer located in the NIC driver and the Traffic Control (TC) layer. This means that we have the opportunity to perform replication functions in high performance by avoiding burdensome kernel networking stack overhead through in-kernel function offloading. Although there are kernel-bypass techniques like DPDK, they are also challenging to deploy because developers should implement complex networking stack functions in userspace. Furthermore, they trade rich kernel properties like security and isolation. Lastly, the kernel-bypass techniques waste CPU resources with constant usage of 100% even at low loads due to polling-based packet reception [11].

We have implemented a Dalio prototype on a testbed consisting of 8 commodity servers. We compare Dalio with a vanilla Linux-based replication coordinator and NetLR [6], a switch-based replication coordinator. Our evaluation results show that Dalio has better throughput than the vanilla Linux by 2.05× and is robust to skewed workloads. In addition, we show that Dalio is comparable to NetLR by providing a sufficient throughput for a replication group.

[†]The author is with the Department of Computer Engineering, Sungshin Women's University, Seoul, Republic of Korea.

a) E-mail: gykim@sungshin.ac.kr (Corresponding author)

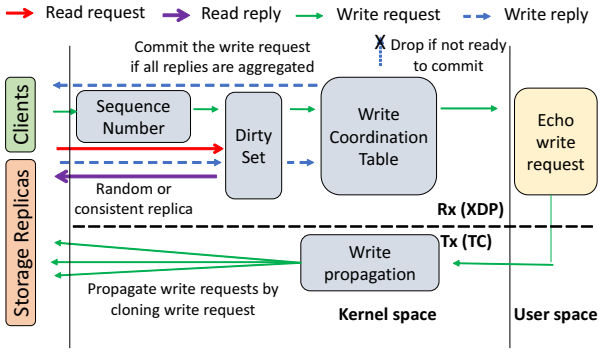

Fig. 1 Dalio architecture and message handling. Read requests/replies and write replies never visit user space, hence the kernel stack overhead is avoided. A write request visits user space since only the TC layer allows packet cloning, but it still improves performance since the packet send function is called only once in user space.

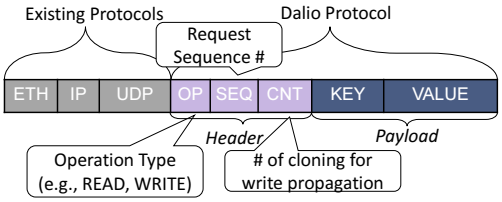

Fig. 2 Dalio packet format. The Dalio message is an L4 payload that does not harm existing network stack functionality.

2. Dalio Design

2.1 Challenges and Solutions

We build an eBPF-based replication coordinator in PSLBs to replicate key-value data in a centralized manner with high performance, linearizability, and deploy-friendliness. However, there are several technical challenges to realize the idea.

First, we should handle multiple pending writes for the same key simultaneously. If pending write requests stay in the PSLB too much for write coordination, subsequent writes for the same item cannot be processed in time because a slot for write coordination is already occupied. This is especially important for software-based centralized replication because the PSLB requires more time to process a packet compared to the switch hardware. Therefore, we separate the dirty set and the write coordination table, where one tracks the key of pending writes and the other performs write coordination based on the request ID instead of the item key. This is because a slot in the dirty set indicates the binary state of an item, which multiple write requests for the same item can share, whereas a slot in the write coordination table should be used by a single request exclusively.

Second, to propagate write requests, we should clone a write request multiple times, but the current eBPF does not provide cloning functionality in the XDP layer. To address this, we leverage `BPF_CLONE_REDIRECT`, the cloning function of the TC layer. To bridge between XDP and TC, we let write packets visit the userspace application that simply forwards the packet upon receiving it. Since typical production workloads like YCSB are read-intensive (e.g., 95:5 for read:write [12]), this does not harm performance much. Additionally, since we only call the packet send function once in the userspace, CPU resources are still saved compared to the traditional approach of calling the send function multiple times for write propagation.

Third, we should minimize memory footprints because eBPF only supports static memory allocation. It means that we may waste memory unnecessarily if memory utilization is low. To address this, our approach allocates small memory space for the dirty set and the write coordination table. Instead, we use multiple hash functions to address hash collisions efficiently. If a collision occurs, Dalio searches for another vacant slot using other hash functions.

Figure 1 shows the high-level architecture of Dalio. Read requests and write replies only visit the XDP layer. Therefore, they can be processed quickly without the networking stack overhead. Write requests visit userspace to utilize the TC layer, but its overhead is small since the application forwards the packets immediately upon receiving them. The TC layer clones the request multiple times for write propagation without the intervention of the userspace application.

2.2 Packet Format

To perform centralized replication, we need some metadata, which should be included in the packet header. Figure 2 illustrates the packet format of Dalio. We use a custom L7 protocol message (i.e., a payload of L4 protocols) consisting of the following fields.

- `OP`: the operation type, which can be `READ`, `WRITE`, `R-REPLY`, `W-REPLY`, `R-REJECT`, and `W-REJECT`.
- `SEQ`: a monotonically-increasing sequence number for request IDs. The PSLB increases this field by one for every write request to distinguish each write request.
- `CNT`: the number of packet cloning that a request experienced. This field is used to propagate write requests to multiple replicas in the TC layer.

2.3 Dalio Modules in XDP and TC Layers

The XDP and TC layers are the core parts of Dalio that handle incoming/outgoing packets. The XDP layer consists of the sequence number, the dirty set, and the write coordination table modules. eBPF supports eBPF maps, which are generic storage of different data types. We use eBPF maps to construct the modules because we should maintain and utilize item states across multiple packets. All three modules use `BPF_MAP_TYPE_ARRAY`, and their roles are as follows.

Sequence number. This module includes a variable for sequencing writes and a lock for concurrent requests. This sequence number acts as a request ID for write coordination.

Dirty set. This module is needed to track the list of inconsistent item keys where write operations are ongoing and not committed yet. With this, read requests for the inconsistent item can be forwarded to a consistent replica. The module contains metadata, such as the item key, the sequence number of the write request, the lock, and the IP and MAC addresses of the latest known consistent replica. Each slot in the module is indexed by the hash of item keys.

Write coordination table. The module counts the number of received write replies and commits the write if all replies are aggregated. Each slot in this module performs write coordination for a single write request. Therefore, a slot is indexed by the sequence number (i.e., request ID). The module includes metadata like the item key, the lock, the number of aggregated write replies, and the client's information (i.e., IP and MAC addresses and the L4 port number).

Write propagation. For the TC layer, we have only one module, the write propagation module. The module replicates the outgoing write request packets as many as the number of replicas to propagate write requests.

2.4 Packet Processing

Read requests and replies. Upon receiving a read request, the PSLB checks whether the requested key is in the dirty set. If it does, for linearizability, the PSLB forwards the request to the latest known replica tracked in the dirty set. Otherwise, the request is forwarded to a random replica. Before forwarding the request, the PSLB copies the stored sequence number in the dirty set to the `SEQ` field in the packet header. This prevents the request from reading the stale data by comparing the carried sequence number and the stored sequence number in the replica (i.e., item version). Read replies do not visit the PSLB, and go directly to the client.

Write requests. When the PSLB receives a write request, it tries to lock the sequence number module for atomic increase operation. For locking, we use `bpf_spin_lock`. If lock is granted, the PSLB increases the sequence number variable by one and assigns the value to the `SEQ` field in the packet header. After that, the lock is released. The PSLB now finds a vacant slot in the dirty set using the key hash of the requested item. As we described, the PSLB uses multiple hash functions to handle hash collisions. If found, the PSLB locks the dirty set slot and checks that the slot is empty or occupied by the same key. When any condition is met, the PSLB puts the item key and the sequence number to the slot. If the PSLB fails to find a matched slot, the request is returned to the client after updating the `OP` field to `W_REJECT`. The client can retransmit the request by receiving the rejected request.

Next, the PSLB tries to lock the slot in the write coordination table after unlocking the dirty set slot. If granted, the PSLB updates the slot with the item key and the client's information (i.e., IP address, MAC address, and L4 port number). To propagate the request, the PSLB lets the packet visit userspace using `XDP_PASS`. The userspace application forwards the packet to the NIC again without modifying the header. The hook at the TC layer captures the packet and clones packets as many as the number of replicas using `BPF_CLONE_REDIRECT`, which creates a copy of the current packet. Each cloning increases the `CNT` field by one. After propagating the write (i.e., reaching `CNT` to the number of replicas), the PSLB finishes handling the write request.

Write replies. The PSLB checks the dirty set to see whether the key is included. If included and the value of `SEQ` field in the packet header is larger than or equal to the stored sequence number, the PSLB updates the IP address in the slot with its source IP address. After that, the PSLB increases the counter variable of the slot in the write coordination table by one. All operations to the dirty set and the write coordination table are done atomically using the `bpf_spin_lock`. If the slot counter is equal to the number of replicas, it means that every replica has updated the write request. Therefore, the PSLB forwards the reply to the client to commit the write. Before forwarding the reply, the slots in the dirty set and the write coordination table are cleared. Otherwise, since we need more replies to commit the write request, the PSLB drops the reply after increasing the slot counter by one.

3. Performance Evaluation

3.1 Methodology

Testbed setup. We build a testbed consisting of 8 commodity servers and a programmable ToR switch for performance evaluation. The servers use an Intel i5-12600K @ 3.7 Ghz, 32 GB of DDR5 memory, and a Nvidia ConnectX-5 100GbE NIC. The servers run Ubuntu 22.04 LTS with Linux kernel 6.5.0. Between the 8 servers, 4 servers act as clients that generate requests. 3 servers are storage servers with replicated data under the same replication group. The other server acts as the PSLB.

Applications and compared schemes. We implement a Dalio prototype on the PSLB using eBPF APIs supported by Linux kernel 6.5.0. For comparisons, we implement the PSLB using a legacy UDP socket (i.e., the vanilla Linux) and NetLR [6] on an Intel Tofino switch using Intel Tofino SDE 9.7.0. We also implement open-loop client and server applications in C with Intel DPDK to maximize the performance of clients and servers. Our key-value stores are based on TommyDS [13], a high-performance data structure library.

Workload. We basically use a read-heavy YCSB workload with 5% of writes [12] and 1M items whose keys and values are 16 bytes and 128 bytes [14]. Since NetLR supports a key size up to 4 bytes, it uses a 4-byte hash for each item key in the switch data plane. We consider both uniform and skewed workloads.

3.2 Results

Throughput with various workloads. We measure the maximum throughput of each scheme with diverse workloads. Figure 3 shows the throughput with different skewness. We can see that Dalio outperforms the vanilla Linux by up to 2.05×. This is because the vanilla Linux causes excessive

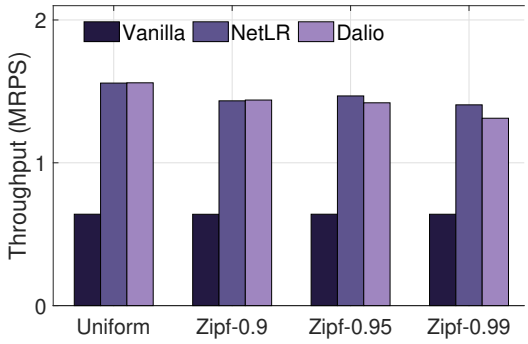

Fig. 3 Throughput with various workloads. Dalio beats the vanilla Linux significantly and shows comparable performance to NetLR.

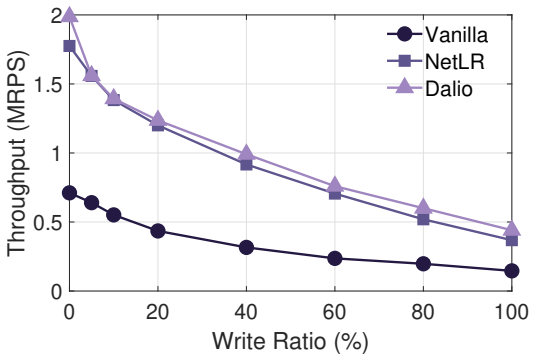

Fig. 4 Impact of write ratios. The gap between NetLR and Dalio is due to the collision between the write requests for the same key in the dirty set of NetLR.

kernel networking stack overhead, whereas Dalio avoids the overhead by handling requests using eBPF. The throughput slightly decreases as the workload becomes more skewed. The performance of Dalio is still comparable to NetLR even with Zipf-0.99 thanks to the eBPF-based request processing.

Impact of write ratios. We now measure the maximum throughput by varying the write ratio with the uniform workload. Figure 4 plots the throughput of the three solutions as the write ratio increases. We can see that the throughput decreases as the write ratio grows. This is not surprising because, unlike the read operation, the write operation involves all replicas, as each replica should maintain the latest data of the item. Regardless of write ratios, Dalio is better than the vanilla Linux. Dalio generally outperforms NetLR slightly. This is because NetLR allows the overwrite of the subsequent write request to the slot where the earlier write request holds when there are concurrent writes for the same item key. This leads to the loss and retransmission of overwritten write requests. Dalio does not experience this collision since we separate the write coordination table and the dirty set.

4. Conclusion

This paper proposed Dalio, a software-based centralized replication system to achieve high performance and linearizability without specialized hardware. We built a replication coordinator in PSLBs using eBPF, an emerging kernel-native technique. We addressed several technical challenges to realize the idea. Experimental results demonstrated that Dalio provides higher throughput than the vanilla Linux by up to 2.05× and is comparable to the state-of-the-art hardware-based solution.

Acknowledgement

This work was supported by the Sungshin Women's University Research Grant of 2024.

References

[1] P. Hunt, M. Konar, F.P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," Proc. of USENIX ATC, USA, p.11, 2010.

[2] A. Katsarakis, V. Gavrielatos, M.S. Katebzadeh, A. Joshi, A. Dragojevic, B. Grot, and V. Nagarajan, "Hermes: A fast, fault-tolerant and linearizable replication protocol," Proc. of ACM ASPLOS, New York, NY, USA, p.201–217, 2020.

[3] P.A. Alsberg and J.D. Day, "A principle for resilient sharing of distributed resources," Proc. of ICSE, Washington, DC, USA, p.562–570, IEEE Computer Society Press, 1976.

[4] R.V. Renesse and F.B. Schneider, "Chain replication for supporting high throughput and availability," Proc. of USENIX OSDI, San Francisco, CA, pp.91–104, Dec. 2004.

[5] J. Terrace and M.J. Freedman, "Object storage on craq: High-throughput chain replication for read-mostly workloads," Proc. of USENIX ATC, p.11, 2009.

[6] G. Kim and W. Lee, "In-network leaderless replication for distributed data stores," Proc. VLDB Endow., vol.15, no.7, pp.1337–1349, Mar. 2022.

[7] "Tofino switch." `https://github.com/barefootnetworks/Open-Tofino`, Last accessed date: 26/06/2024, 2023.

[8] Z. Cao, S. Dong, S. Vemuri, and D.H. Du, "Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook," Proc. of USENIX FAST, Santa Clara, CA, Feb. 2020.

[9] "ebpf." `https://ebpf.io/`, Last accessed date: 26/06/2024, 2024.

[10] M. Primorac, K. Argyraki, and E. Bugnion, "When to hedge in interactive services," Proc. of USENIX NSDI, pp.373–387, April 2021.

[11] Y. Zhou, X. Xiang, M. Kiley, S. Dharanipragada, and M. Yu, "DINT: Fast In-Kernel distributed transactions with eBPF," Proc. of USENIX NSDI, Santa Clara, CA, pp.401–417, USENIX Association, April 2024.

[12] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," Proc. of ACM SoCC, New York, NY, USA, p.143–154, Association for Computing Machinery, 2010.

[13] "Tommyds c library." `https://www.tommyds.it/`, Last accessed date: 26/06/2024, 2018.

[14] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," Proc. of ACM SOSP, pp.121–136, 2017.