ALOR: Adaptive Layout Optimization of Raft Groups for Heterogeneous Distributed Key-Value Stores

Yangyang Wang, Yunpeng Chai, Xin Wang

ALOR: Adaptive Layout Optimization of Raft Groups for Heterogeneous Distributed Key-Value Stores

Yangyang Wang[1,2], Yunpeng Chai[1,2*], and Xin Wang[3]

[1]Key Laboratory of Data Engineering and Knowledge Engineering, MOE
[2]School of Information, Renmin University of China, Beijing, China
[3]College of Intelligence and Computing, Tianjin University, Tianjin, China
*Corresponding Author: ypchai@ruc.edu.cn

Abstract. Many distributed key-value storage systems employ the simple and effective Raft protocol to ensure data consistency. They usually assume a homogeneous node hardware configuration for the underlying cluster and thus adopt even data distribution schemes. However, today's distributed systems tend to be heterogeneous in nodes' I/O devices due to the regular worn I/O device replacement and the emergence of expensive new storage media (e.g., non-volatile memory). In this paper, we propose a new data layout scheme called *Adaptive Layout Optimization of Raft groups* (ALOR), considering the hardware heterogeneity of the cluster. ALOR aims to optimize the data layout of Raft groups to achieve a better practical load balance, which leads to higher performance. ALOR consists of two components: *leader migration in Raft groups* and *skewed data layout based on cold data migration*. We conducted experiments on a practical heterogeneous cluster, and the results indicate that, on average, ALOR improves throughput by 36.89%, reduces latency and 99th percentile tail latency by 24.54% and 21.32%, respectively.

1 Introduction

Due to the excellent scalability and efficiency, key-value (KV) stores have been widely adopted by many big data systems (e.g., Cassandra and HBase). Many distributed KV storage systems employ the Raft [1] protocol to ensure data consistency because it is easy to be implemented in practical systems. These distributed KV systems coupled with Raft are usually designed for homogeneous systems. However, today's distributed systems tend to be heterogeneous, especially for nodes' I/O devices. The reason lies in the following two aspects:

- The annual disk replacement rates in large-scale distributed systems are typically 2-4% and can be up to 13% in some systems [2]. The replacement rates of Solid State Drives (SSDs) are usually higher than disks due to the limited write endurance of Flash chips. That is to say, in a large-scale distributed KV system, I/O devices are regularly replaced with new generations of I/O products, and these new products usually have higher performance and cost-efficiency than the old ones.
- The emerging storage devices (e.g., SSDs or non-volatile memory (NVM) [3]) have obvious performance advantages over the traditional ones. However, these new devices are usually much more expensive, so we usually deploy them in only a subset of the clusters for cost efficiency.

In distributed storage systems, the Raft protocol is usually adopted to ensure data consistency by defining the different behaviors of the only leader and the other followers for the same data segment. In consequence, the Raft protocol has the inherent heterogeneous feature, i.e., the leader in a Raft group usually takes more jobs and has greater impact on the performance than the followers do. In a heterogeneous distributed KV storage system based on the Raft protocol, if many leaders locate on slow nodes, the performance of the entire system will be slowed down, because the result is not returned to the client until the corresponding leader completes applying the log into the data set (see Section 2.1 for more details). Considering this feature of Raft, the hardware heterogeneity is not necessarily a negative factor. Instead, if we can adapt heterogeneity of Raft to the hardware heterogeneity of distributed KV systems through data layout optimization of Raft groups, the system performance can be improved.

In this paper, we propose a new scheme called *Adaptive Layout Optimization of Raft groups* (ALOR) to match the data layout with the hardware heterogeneity of distributed KV systems for higher performance. ALOR consists of two components: *leader migration in Raft groups* (Section 3.1) and *skewed data layout based on cold data migration* (Section 3.2). The experiments based on a practical heterogeneous cluster indicate that, on average, ALOR improves throughput by 36.89%, reduces the average latency by 24.54%, and reduces 21.32% tail latency. Furthermore, if we construct hybrid devices with two kinds of different devices (e.g., NVM and SSDs) on each node of the cluster, ALOR can still achieve a 28.57% higher write throughput compared with this homogeneous hybrid device solution coupled with the same hardware resources.

The rest of this paper is organized as follows. Section 2 introduces the background and related work. The detailed design of our proposed ALOR is presented in Section 3, followed by the evaluations in Section 4. Finally, Section 5 concludes this paper with a summary of our contributions.

2 Background and Related Work

2.1 The Raft Protocol

Traditionally, Paxos [4] is a classical protocol to ensure data consistency in distributed systems. However, Paxos was particularly difficult to understand and implement. In this case, the Raft protocol [1], which is readily comprehensible and realized, has been quickly adopted by many practical distributed systems like Etcd [5], TiKV [6], and PolarDB [7] since it was proposed in 2014.

According to Raft, the main process of serving read and write requests can be found in Fig. 1. We assume a Raft group contains three copies located in three different nodes in the cluster, i.e., one and only one elected leader and two followers.

When a write request arrives at the leader from users, the leader both appends the new contents to the local log and forwards them to the two followers. After more than half of the nodes (i.e., two in this case, including the leader itself) have accomplished the logging action successfully, the leader will proceed to apply the request log, i.e., insert/update the new data into the structured

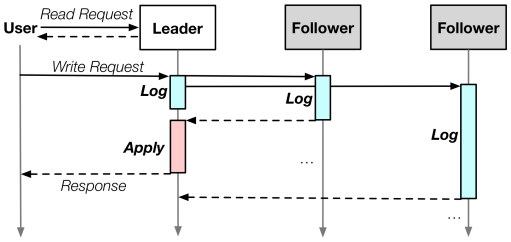

Fig. 1. The main process of serving requests according to Raft.

key-value store. Then, the user can get the response of this write request from the leader.

In addition, all read requests are served by the leader alone to ensure the data consistency. In order to ensure linear consistency, Raft will ensure that all the previous logs have been applied before the read request is served.

2.2 Related Work

Raft/Paxos Improvements. In order to reduce the high latency of the Paxos protocol, Wang et al. proposed APUS [15], the first RDMA-based Paxos protocol that aims to be fast and scalable to client connections and hosts. PolarFS [16] implements a parallel Raft to allow parallel submission of logs, breaking Raft's strict limitation that log has to be continuous, with the benefit of increasing concurrency. In order to reduce the latency of distributed systems, Guerraoui et al. proposed Incremental Consistency Guarantees (ICG) [17]. In addition, Alagappan et al. [18] proposed correlated crash vulnerabilities to ensure data security in distributed systems.

Heterogeneous Systems. Zhang et al. developed Mega-KV [19], a high-performance distributed in-memory key-value store system on a heterogeneous CPU-GPU cluster. Dey et al. [20] proposed an approach that gives multi-item transactions across heterogeneous data stores. Strata [21] and OctopusFS [22] designed file systems for heterogeneous storage devices on a single node.

Therefore, few research works consider the heterogeneous I/O performance among nodes in a cluster. This paper will focus on the performance optimization in heterogeneous distributed key-value storage systems.

3 The Design of ALOR

In this section, we will present the detailed design of our proposed *Adaptive Layout Optimization of Raft groups* (ALOR) scheme, which aims to improve the performance of distributed key-value storage systems in case of heterogeneous situations. The two main components of ALOR will be introduced in Section 3.1 and Section 3.2, respectively.

3.1 Leader Migration in Raft Groups

According to the Raft protocol, the performance of service nodes does not affect the leader election. In this case, the leader and the followers in a Raft group are usually randomly and evenly distributed among all the service nodes for the sake of load balance no matter the underlying system is homogeneous or heterogeneous. For example, as Fig. 2 shows, we assume that there are four Raft

groups and each group contains three copies of data in a distributed KV storage system with six nodes. According to the original Raft protocol, the data blocks and the leaders are evenly distributed.

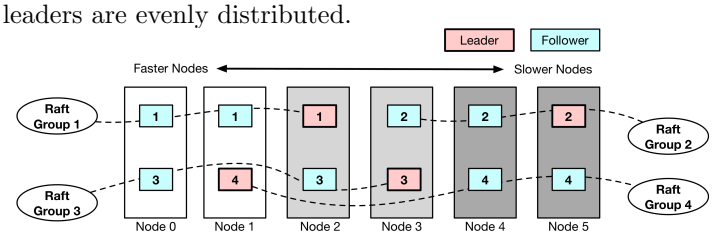

Fig. 2. Raft groups are usually evenly distributed among nodes for load balance.

However, the leader in a Raft group plays the most important role in affecting the performance (e.g., users always read data from leaders and write requests are not confirmed until the leader applies the log). If the leader is placed on a slow node, the performance of accessing this Raft group will be slowed down. Therefore, ALOR gradually migrates leaders to the node with the best performance in Raft groups. The larger the performance gap among the nodes in a Raft group is, the higher priority of migration the corresponding leader will be given in ALOR, as illustrated in Fig. 3. Furthermore, as long as a follower catches up the same status of logging and applying data as the leader, it can be easily set as the new leader with negligible overhead.

For write operations, the leaders and the followers perform the same work (i.e., writing the log first and then applying it). In this case, although the nodes with higher performance undertake more leaders, their average loads are the same as each other. In fact, ALOR just fully utilizes the fast processing of high-performance nodes in a heterogeneous system to reduce the process time of users' write requests.

For read operations, according to ALOR, the nodes with high performance usually store more leaders than the nodes with low performance. Because users always read data from leaders for strong consistency, the high-performance nodes serving most read requests can reduce the response time of read request processing in most cases. Although the high-performance nodes will undertake more read requests, the read request processing is much more lightweight than the write requests in a key-value storage system due to the significant write amplification of the KV indexes (e.g., B-tree, LSM-tree, etc.).

For a read-write mixed workload, write operations will slow down read operations, because Raft ensures linear consistency, i.e., read operations must be performed after all the previous write operations have been completed. So speeding up the write operations is critical for improving system performance.

3.2 Skewed Data Layout based on Cold Data Migration

Skewed Data Layout. The idea of promoting system performance in a heterogeneous distributed store is to put appropriate load on nodes according to their ability. Although the aforementioned leader migration mechanism in ALOR puts more leaders on the strong nodes, this is not enough. We should further optimize the data amount distribution during the disk-filling process, i.e., putting more

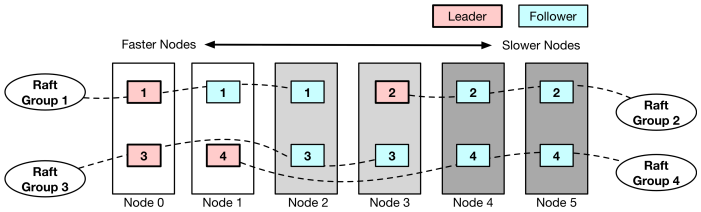

Fig. 3. ALOR migrates the leaders in Raft groups to the faster node as far as possible for higher performance.

data on the high-performance nodes. The skewed data layout in ALOR can fully utilize the fast processing speed of the high-performance nodes for higher system performance.

However, it also causes two issues: (1) How to set an appropriate data-filling speeds according to the performance of a node? (2) Assuming all nodes have the same storage capacity, some high-performance nodes will be full ahead of others due to the different data-filling speed setting. Thus how to process the new arrival data after some nodes are full is a problem. The solutions to these two problems in ALOR will be presented in the following parts.

Disk-Filling Speed Setting. In ALOR, the disk-filling speed of nodes is set to be proportional to the average performance of writing key-value pairs into the KV store in the node. For example, shown as Fig. 4, assuming there are six nodes and their KV accessing performance is 3:3:2:2:1:1, the proportion of data that they get is similar to this ratio. In this case, the load on a node matches its key-value pairs processing ability.

The next problem is how to estimate the key-value accessing performance of the nodes. The difficulty lies in that a distributed key-value store usually does not supply a KV accessing interface on a single node. Our solution is to automatically measure the I/O performance of a node during its initialization process by calling tools like *fio* [23]. However, the I/O performance is not linear with the node's KV accessing performance. Thus we measured both the I/O and the KV performance of several representative nodes and construct their relationship beforehand. Then we can fit the KV performance of the nodes through their measured I/O performance (See Fig. 10 in the experimental part for reference).

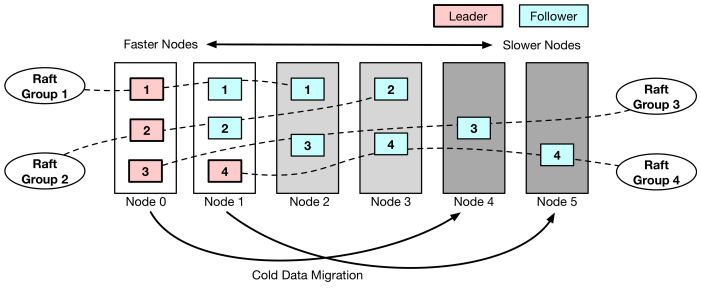

Fig. 4. The distribution of Raft groups in ALOR.

Cold Data Migration. In ALOR, the skewed data layout is achieved through the specially designed data migration mechanism. An important weight, i.e.,

Data Weight, is employed to control the data migration among nodes. The condition of migrating some data in node A to node B can be expressed as Eq. 1, where S_A and S_B are the data volume of node A and B respectively, and S_M is the size of the to-be-migrated data.

$$\frac{S_A - S_M}{DataWeight_A} > \frac{S_B + S_M}{DataWeight_B} \tag{1}$$

If the nodes A and B have the same data weights, Eq. 1 aims to balance the stored data amount between them through data migration. In a heterogeneous system, the strong nodes should have larger data weight values to undertake more data and more requests than weak nodes. In order to reach the above disk-filling speed setting, the data weight values of nodes can be set according to their key-value accessing performance.

When the data volume of a node reaches a specified threshold (e.g., 95% of its capacity), we need to migrate some cold data in this node to others, thus making room for the new arrivals. Then the node's data weight will be set to a very small value (e.g., 10^{-6}), some of its cold data will be migrated to other nodes. When its data volume is lower than the threshold again, the data migration of this node is stopped, avoiding introducing too much overhead.

The advantage of the cold data migration mechanism in ALOR is to promote the hotness of the stored data in high-performance nodes (e.g., Node 0 in Fig. 4), whose side-effect lies in the additional overhead of data migration among nodes. However, the overhead of migrating data is small, because sequential read and write operations of key-value pairs are performed during the data migration process, which are much faster than random GET/PUT operations from users.

4 Implementation and Evaluation

We implemented ALOR based on TiDB [8], one of the most widely used open source NewSQL databases similar to Google Spanner [9]. TiDB is mainly composed of three projects: TiDB (i.e., the SQL Layer), TiKV (i.e., a distributed key-value storage system based on Raft), and the Placement Driver (PD), which is the managing component of the cluster. PD consists of 480K LOC of *Go* and TiKV consists of more than 84K LOC of *Rust*. TiKV has become one of the largest open source projects in the *Rust* community. To implement ALOR, we have added 200+ LOC of *Rust* in TiKV and 400+ LOC of *Go* in PD. The source codes of our implementation of ALOR are on Github now (*https://github.com/vliulan/ALOR*).

4.1 Experimental Setup

We will compare our proposed ALOR scheme with the widely used scheme which evenly distributing (ED) all the data and leaders of Raft groups in distributed systems. The experiments were performed in a cluster of eight physical nodes; each of them is coupled with Linux Centos 7 3.10.0, 16GB DRAM and a 16-GB non-volatile memory (NVM) block device, where NVM is emulated by DRAM. Nodes can be equipped with two kinds of Solid State Drives (SSDs), i.e., a 280GB version of Intel Optane 900p PCIe SSD (a.k.a, high-end SSD) or a 256GB Intel

SATA SSD (a.k.a, plain SSD). Six of the nodes serve as TiKV node, one node as PD, and one node runs the benchmark tool, i.e., go-YCSB [10].

Go-YCSB is a *Go* language version of the widely used YCSB benchmark [11]. In the experiments, the workloads we selected include *Load* (insert-only), *Workload A* (50:50 read/update), *Workload B* (95:5 read/update), and *Workload C* (read-only) of YCSB. Other configurations of the workloads can be found in the specification [12]. Each key-value pair contains a 16-B key and a 1-KB value, and each data block has three copies in TiKV. Although the performance of the storage devices is heterogeneous, the data capacities of all the TiKV nodes are set to be the same (5GB by default).

In the following experiments, we adopt the system throughput (operations per second, i.e., ops/sec), the average latency, and the 99th percentile latency to evaluate the system performance.

4.2 Overall Results

In the overall experiments, among the six TiKV nodes in the cluster, one node equips the fastest NVM block device, two node equip the high-end SSDs, and the slowest plain SSDs are deployed in the other three nodes. We first load 10GB of data to fill the cluster (i.e., 30GB data considering the replicas), and then perform workloads A, B, and C, respectively, accessing 10GB of data each.

As Fig. 5 plots, ALOR achieves higher throughput than ED in most cases, i.e., 72.6% higher in *Load*, 61.5% higher in *Workload A*, and 13.7% higher in *Workload B*. On average, ALOR promotes the throughput by 36.89% . Compared with the traditional even distribution (ED) solution, which is appropriate in homogeneous distributed systems, ALOR puts properly more data and more leaders on the fast nodes according to nodes' heterogeneous ability. In fact, the practical load balance of a heterogeneous system is improved coupled with ALOR, leading to a higher throughput.

For read operations, ALOR concentrates more leaders, which serve all the read requests, on fast nodes. The benefit is to boost the processing of read requests; the disadvantage lies in that when the load of fast nodes is too high, some requests have to wait a moment. So in the read-only *Workload C*, the throughput of ALOR is a bit lower than, but very close to ED.

For write operations, ALOR certainly boosts the request processing. The reason lies in two aspects: (1) Since a leader has to log and apply the written data before replying the user, the faster nodes can boost these actions of leaders. (2) More than half nodes have to log the written data before replying the user, and more data segments (leader or follower) in a Raft group have the possibility to locate on faster nodes because of the skewed data layout in ALOR.

Fig. 6 and 7 exhibit the results of the average latency and the 99th percentile latency. One average, ALOR reduces the latency by 24.54% compared with ED. The average read latency improvement of ALOR in *Workload C* is slightly larger than ED, but those of ALOR *Workload A* and *Workload B* are smaller than ED, because reducing the write processing time leads to less waiting time of read operations in read-write mixed workloads. For both read and write operations, ALOR reduces the tail latency, i.e., 21.32% on average compared with ED.

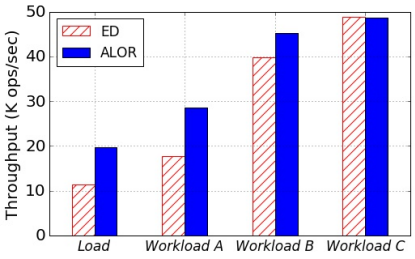

Fig. 5. Overall throughput results.

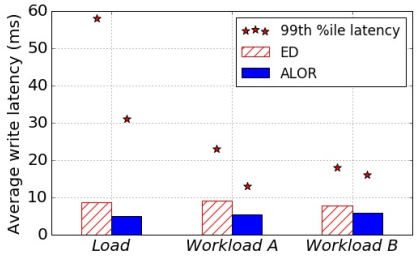

Fig. 6. Overall write latency results.

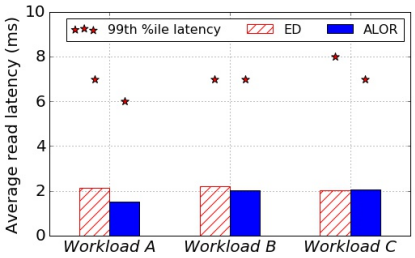

Fig. 7. Overall read latency results.

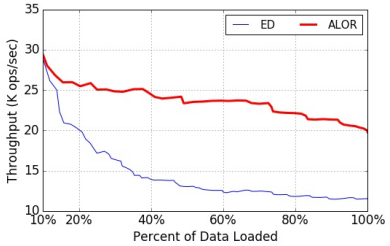

Fig. 8. Throughput during the data loading process.

Fig. 8 and 9 plot the changes of throughput and latency during the data loading process. In the very beginning, three copies of data are written into the three fast nodes first for both ALOR and ED, so the performance of the front ALOR and ED is almost the same. Then, the performance of ALOR and ED both decreased due to the filled cache, but ED dropped more. The overall performance of ALOR is much higher than ED during the whole data loading process in all the aspects of throughput, average latency, and tail latency.

4.3 KV Performance Estimation

Recall Section 3.2 that we estimate key-value accessing performance according to I/O performance. The KV engine used by TiKV is RocksDB [13], a famous open source KV engine based on LSM tree developed by Facebook. The granularity of RocksDB writing is megabytes (e.g., 8MB). Therefore, we first utilize *fio* to measure the I/O performance of randomly writing 8-MB blocks.

We selected three typical devices: NVM block device (emulated by DRAM), high-end SSD, and plain SSD in the measurements, and performed multiple single-point I/O performance tests based on *fio* and KV performance tests based on go-YCSB and RocksDB on the single node. Then we can build the estimated relationship between the two factors through polynomial function fitting.

The measured I/O and KV performance results and the curve of the fitted function are shown in Fig. 10. Taking the red box in the figure as an example, if the disk performance measured by fio is 2GB/s, we can estimate the nodes' KV performance as 22MB/s.

4.4 Impacts of Different Heterogeneous Configurations

In this part, we will evaluate ALOR under different heterogeneous configurations, including two high-end SSDs and four plain SSDs (i.e., 2H4P), one NVM block device, two high-end SSDs and three plain SSDs (i.e., 1N2H3P), two NVM

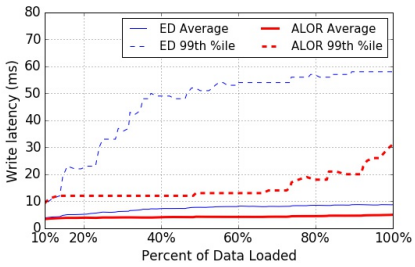

Fig. 9. Latency during the data loading process.

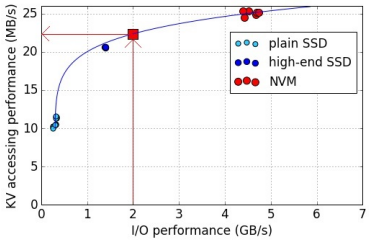

Fig. 10. Relationship between I/O and KV performance.

block devices, two high-end SSDs and two plain SSDs (i.e., 2N2H2P), and three NVM block devices, two high-end SSDs and one plain SSD (i.e., 3N2H1P). For different settings, we all loaded 10GB data into the cluster to measure the system throughput, latency, and tail latency of ALOR and ED, as shown in Fig. 11 and 12, respectively.

The performance of ALOR is improved compared with ED, but the 2N2H2P and 3N2H1P configurations' enhancements are not as much as the other two. The reason lies in that the heterogeneous situations in the 2N2H2P and 3N2H1P configurations are not as significant as the other two ones.

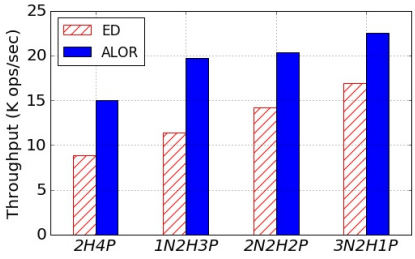

Fig. 11. Throughput under different heterogeneous configurations.

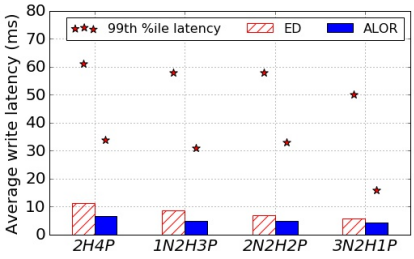

Fig. 12. Latency under different heterogeneous configurations.

4.5 Impacts of System Scale

In order to evaluate the scalability of ALOR, we performed experiments on clusters with different counts of TiKV nodes (i.e., 4, 5, or 6 TiKV nodes). The configuration of the 4 TiKV nodes is one high-end SSD and three plain SSDs, that of the 5 TiKV nodes is one NVM block device, one high-end SSD, and three plain SSDs, and the configuration of the 6 TiKV nodes is one NVM block device, two high-end SSDs, and three plain SSDs.

For the 6 TiKV nodes, we wrote 10GB data into the cluster; proportionally, we wrote 8.33GB data into the 5 TiKV nodes, and 6.67GB data into the 4 TiKV nodes. The throughput and latency results of ALOR and ED are shown in Fig. 13 and 14, respectively. As the cluster's node count increases, the performance of ALOR and ED both increase. ALOR exhibits stable performance advantage compared with ED under various system scales.

4.6 Analysis of ALOR Components

Recall Section 3 that ALOR has two components, i.e., the leader migration and the skewed data layout based on cold data migration. In this part, we will

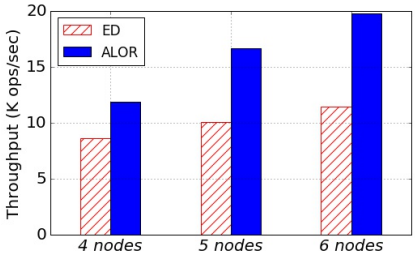

Fig. 13. Throughput under different system scales.

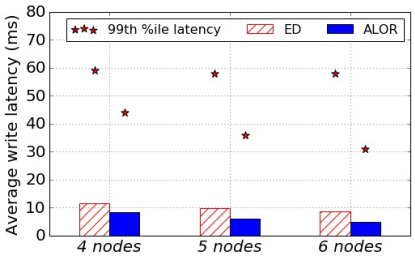

Fig. 14. Latency under different system scales.

evaluate how much the two components of ALOR contribute on the performance improvement. Therefore, we constructed a special version of ALOR with only the leader migration module, i.e., Leader Migration Only (LMO). The comparison among ED, LMO, and ALOR can show us the performance contributions of ALOR's two components.

As Fig. 15 and 16 plot, we first load (insert-only) 10GB data to fill the cluster, and then perform *Workload C* (read-only) by reading 10GB data. The experimental results show that the load performance of LMO is 22.85% higher than ED and ALOR is 40.53% higher than LMO. That means within the 72.64% throughput improvement of ALOR compared with ED, the leader migration module contributes about 31.45% of it, while the skewed data layout contributes about 68.55%. The average latency and the tail latency of writing are both improved by ALOR's two modules.

The read throughput and average latency of ED, LMO and ALOR are very close to each other, indicating the two modules of ALOR both do not affect the read performance much.

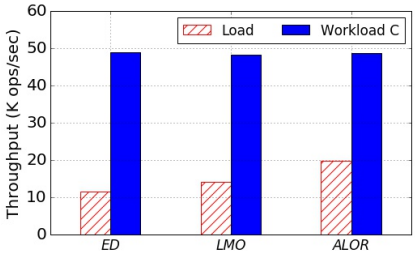

Fig. 15. Throughput comparison among ED, LMO, and ALOR.

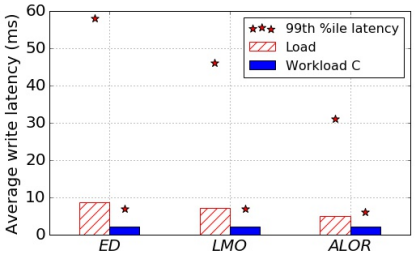

Fig. 16. Latency comparison among ED, LMO, and ALOR.

4.7 ALOR vs. Homogeneous Hybrid Device Solution

When both fast storage devices and slow devices are deployed in a distributed system, an alternative solution is to distribute the fast devices evenly among all the nodes and to construct hybrid devices, in which a fast device acts as the cache of a slow device. In this case, although different devices are in the system, the resources and configurations on each node are homogeneous. We use Flashcache [14] to combine NVM block devices and plain SSDs into hybrid devices on each node. The homogeneous hybrid device solution consumes exactly the same resources as ALOR.

In this part, the experiments were performed on 4 TiKV nodes. Both ED and ALOR are deployed in a cluster with one NVM device and three plain SSDs, each of which can hold up to 5GB data. For the Flashcache solution, it requires four plain SSDs and 4 NVM devices. In order to guarantee the fairness, each plain SSD for Flashcache can only store 3.75GB data (i.e., 3*5GB/4), and each NVM device can hold 1.25GB data (i.e., 5GB/4).

We first loaded 5GB data into the cluster (i.e., 15GB including replicas), and then performed *Workload C* to read 5GB data. The experimental results are shown in Fig. 17 and 18. Although the Flashcache solution achieves higher write performance compared with ED due to better utilization of fast devices, the write throughput of ALOR is 28.57% higher than Flashcache. This indicates that ALOR coupled with heterogeneous node performance configuration is more appropriate for Raft than the homogeneous hybrid device solution.

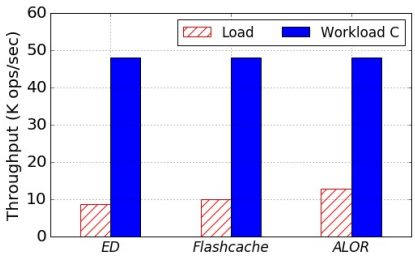

Fig. 17. Throughput comparison among ED, Flashcache, and ALOR.

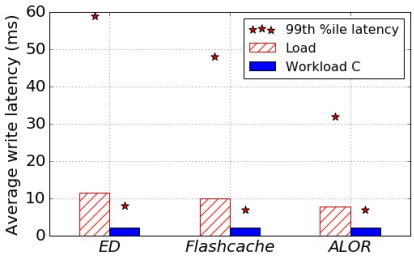

Fig. 18. Latency comparison among ED, Flashcache, and ALOR.

5 Conclusion

In this section, we conclude this paper with a summary of our contributions:

(1) We found and verified that by matching the inherent heterogeneity of Raft groups and the hardware heterogeneity of distributed key-value stores, the system performance could be promoted.

(2) We proposed a new optimized data layout scheme called ALOR, which achieves an appropriate layout of data and Raft leaders in a heterogeneous distributed key-value storage system through *the leader migration* and *the skewed data layout mechanisms*.

(3) The experiments based on a practical heterogeneous cluster indicate that ALOR can promote the write throughput by up to 72.6% than the even data distribution solution, while achieving similar read performance.

Acknowledgement

This work is supported by the National Key Research and Development Program of China (No. 2018YFB1004401), National Natural Science Foundation of China (No. 61732014, 61472427, and 61572353), Beijing Natural Science Foundation (No. 4172031), the National Science Foundation of Tianjin (17JCYBJC15400), the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China (No. 16XNLQ02), and open research program of State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Science (No. CARCH201702) .

References

1. Diego Ongaro, John Ousterhout.: In Search of an Understandable Consensus Algorithm. USENIX Annul Technical Conference(2013)
2. Schroeder B, Gibson G A.: Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? FAST, 2007, 7(1):1-16
3. Wikipedia.: Non-volatile memory. https://en.wikipedia.org/wiki/Non-volatile_memory (2018)
4. Leslie Lamport.: Paxos Made Simple. ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)
5. CoreOS.: Etcd Documentation. http://etcd.readthedocs.io/en/latest (2018)
6. PingCAP.: TiKV. https://github.com/pingcap/tikv (2018)
7. Alibaba Cloud.: PolarDB. https://www.alibabacloud.com/campaign/polardb-discount-icde-2018?spm=a2c5t.10695662.1996646101.searchclickresult.53f66bd8ztuvrS
8. TiDB. PingCAP.: TiDB. https://github.com/pingcap/tidb (2018)
9. James C. Corbett, Jeffrey Dean, et al.: Spanner: Googles Globally-Distributed Database. Acm Transactions on Computer Systems, 2012, 31(3):8
10. PingCAP.: go-ycsb. https://github.com/pingcap/go-ycsb (2018)
11. Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears.: Benchmarking cloud serving systemswith ycsb. In ACM Symposium on Cloud Computing, pages 143154, (2010)
12. PingCAP.: workloads. https://github.com/pingcap/go-ycsb/tree/master/workloads (2018)
13. Facebook.: RocksDB. http://rocksdb.org/ (2018)
14. Facebook.: Flashcache. https://wiki.archlinux.org/index.php/Flashcache (2018)
15. Cheng Wang, Jianyu Jiang, Xusheng Chen, Ning Yi, and Heming Cui.: APUS: Fast and Scalable Paxos on RDMA. In Proceedings of SoCC 17, Santa Clara, CA, USA, September 2427, 2017, 14 pages
16. https://www.alibabacloud.com/blog/deep-dive-on-alibaba-clouds-next-generation-database_578138
17. Rachid Guerraoui, Matej Pavlovic, and Dragos-Adrian Seredinschi.: Incremental Consistency Guarantees for Replicated Objects. In the Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)
18. Ramnatthan Alagappan, Aishwarya Ganesan, Yuvraj Patel, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau.: Correlated Crash Vulnerabilities. In the Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)
19. Kai Zhang, Kaibo Wang, Yuan Yuan, Lei Guo, Rubao Li, Xiaodong Zhang, Bingsheng He, Jiayu Hu, Bei Hua.: A distributed in-memory key-value store system on heterogeneous CPU-GPU cluster. The VLDB Journal (2017) 26:729750
20. Akon Dey, Alan Fekete, Uwe Rohm.: Scalable Transactions across Heterogeneous NoSQL Key-Value Data Stores. The 39th International Conference on Very Large Data Bases(2013)
21. Youngjin Kwon, Henrique Fingler, Tyler Hunt, Simon Peter, Emmett Witchel, Thomas Anderson.: Strata: A Cross Media File System. ACM Symposium on Operating Systems Principles(2017)
22. Elena Kakoulli, Herodotos Herodotou.: OctopusFS: A Distributed File System with Tiered Storage Management. ACM Conference on Management of Data(2017)
23. Jens Axboe. Flexible I/O Tester.: https://github.com/axboe/fio