Enabling a Network Key-Value Store with a Key-Value Storage Interface Using the SPDK

Yeohyeon Park[1], Chang-Gyu Lee[1], Kyungpyo Kim[2], Sung-Soon Park[2], Youngjae Kim[1*]
[1]Sogang University, Seoul, South Korea. [2]Gluesvs

1 INTRODUCTION

A key-value store fundamentally stores and retrieves values using a key-based lookup mechanism instead of a traditional Logical Block Address (LBA) interface. Numerous key-value storage/database platforms such as RocksDB, MongoDB, and Cassandra, despite implementations for efficient key-value stores, still require avoiding heavy software layer overheads(e.g., file system in the OS). Consequently, recent implementations of key-value SSDs [2, 3] provide low I/O latency and high throughput. Despite the superiority of the key-based lookup mechanism, on the other hand, in a disaggregated storage environment, network storage such as SAN and NAS lacks research to adopt a key-value interface, so it is still saddled heavy software layer overheads. Therefore, in this study, we design a network key-value store that has a key-value interface, completely avoids the OS's file system, and allows clients to access <key, value> tuples on the computer network as if they were local key-value storage. We finally reveal its implementation feasibility.

2 PROPOSED DESIGN

Figure 1 shows various structures of software stacks in a distributed database built on various disaggregated storage architectures. Note that the underlying storage interfaces with the DB in diverse ways through a key-value store. In Figure 1(a)(b), the server exports disk volumes or file systems to clients, stacking the file system and key-value store on top. Either way, a client's key-value request inevitably entails the OS's file system overhead. Figure 1(c)(d) describes our proposed architectures. Both adopt Intel SPDK [1]-based key-value store implementations (SPDK-based KVS), eliminating the OS's file system from the software stack. In Figure 1(c), each client runs SPDK-based KVS on the mounted volume. So, key-value requests are processed on the client-side, but the LBA-based NVMe-oF protocol is still used between the remote volume and the client. Unlike Figure 1(c), Figure 1(d) runs SPDK-based KVS on the server and uses the extended NVMe-oF protocol for KV interface support (KV NVMe-oF protocol). Comparing Figure 1(c) and (d), the server-side implementation of SPDK-based KVS offers higher ease of use, manageability, and data sharing of key-value stores than the client-side implementation.

Implementation: We took the server-side approach (Figure 1(d)) for preliminary evaluation of SPDK-based KVS. We implemented the SPDK-based KVS using a simple hash data structure to index <key, value> tuples. The hash table manages corresponding LBAs for the key, and the bitmap arrays keep track of the free logical blocks in the storage.

Operation Flow for I/O: The client-side application sends a key-value request to SPDK-based KVS running on the server via the KV NVMe-oF protocol. Ahead of all, SPDK-based KVS runs the hash function on the received key. For Put(), SPDK-based KVS refers to the bitmap array (for the first write) and allocates the space for the value. Once the value is written to allocated LBAs then the bitmap and hash table are updated accordingly. At last, SPDK-based KVS sends the completion messages back to the client then the NVMe driver on the client-side notifies the application of its completion. For Get(), SPDK-based KVS gets the LBAs of the requested key by referring to the hash table for the key's hash value. Then, the value is read via ordinary NVMe block read. Note that SPDK employs a userspace NVMe driver; thus, both operations are also handled in the userspace.

[*]Y. Kim is the corresponding author.

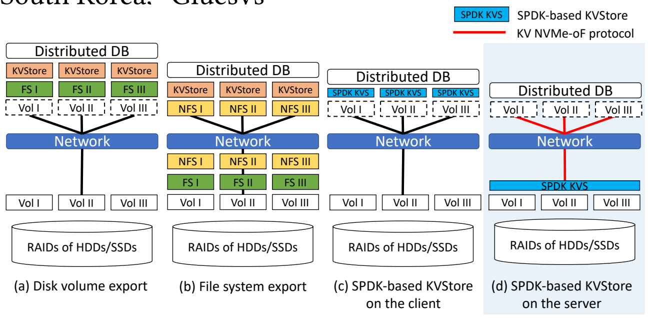

Figure 1: A description of software stacks in a distributed database built on various disaggregated storage architectures.

3 EVALUATION

We used two servers connected via 10 Gbps Ethernet, each equipped with 10 cores and running Linux. Samsung 970 EVO 500 GB NVMe SSD was used on the storage server. SPDK-based KVS was implemented using SPDK v21.10. We evaluated SPDK-based KVS compared to the baseline as shown in Figure 1(a) where RocksDB runs upon the EXT4 file system. Both architectures are tested with the write workload "Fillsequential" with a 32 KB value of RocksDB's *db_bench*.

Table 1: Evaluation Results

Put()	Latency	99th Percentile	Throughput
Baseline	1,835 us	21,224 us	255 MB/s
SPDK-based KVStore	735 us	1,853 us	638 MB/s

We found that SPDK-based KVS reduced average latency by 60%, tail latency by 91% for 99th percentile, and increased throughput by 183%.

Acknowledgement: This research was funded by the Institute of Information and Communications Technology Planning and Evaluation (IITP), Korea government (MSIT) under Grant 2020–0–00104.

REFERENCES

[1] 2022. SPDK. https://spdk.io/.
[2] C. Lee et. al. 2019. iLSM-SSD: An Intelligent LSM-Tree based Key-Value SSD for Data Analytics. In *MASCOTS '19*.
[3] J. Im et. al. 2020. PinK: High-speed In-storage Key-value Store with Bounded Tails. In *USENIX ATC '20*.