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]Dept. of Computer Science and Engineering, Sogang Universitym, Seoul, South Korea
[2]Gluesys
{yeohyeon, changgyu, youkim}@sogang.ac.kr, {kpkim, sspark}@gluesys.com

Key-Value Store Limitations in A Disaggregated Storage Environment

- Key-value stores (KVStore/KVS) are widely used due to its simple key-based lookups to store and retrieve large amounts of data.
- However, research on adopting key-value interfaces in a disaggregated storage environment is still lacking.
- We found that when applying key-value storage to network storage such as SAN and NAS, the overhead of heavy software layers such as the OS's file system is inevitably unavoidable.

Various Software Stacks of A Distributed Key-Value Platform

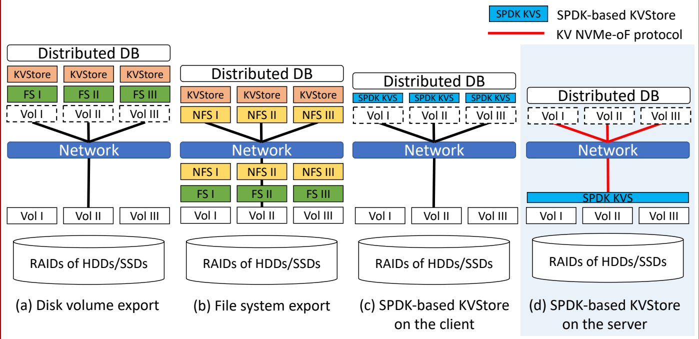

Figure 1. < Software stacks of a distributed key-value platform >

- Figure 1(a)(b) depicts the software stacks of a distributed key-value platform on a SAN or NAS, respectively.
- Figure 1(a) exports the server's disk volume to the client and stack the file system and key-value store.
- Figure 1(b) exports the server's file system to the client via NFS and stacks a key-value store on top of it.
- Either way, a key-value request from a client inevitably entails the OS's file system overhead.
- Figure 1(c)(d) describes our proposed architectures, where both adopt Intel SPDK-based key-value store implementations (SPDK-based KVStore).
- We chose Figure 1(d) because it provides higher easy of use, manageability, and data sharing through centralization of the key-value store.

Proposed Design and Implementation

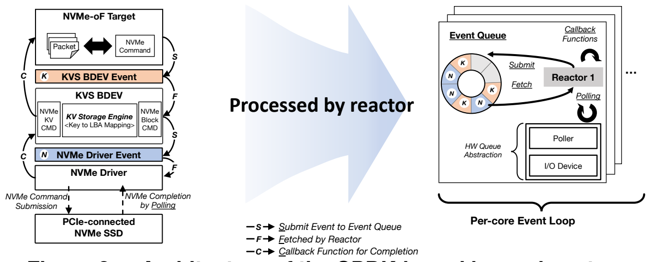

Figure 2. < Architecture of the SPDK-based key-value store implementation (KV BDEV) in the server (Figure 1(d)) >

- We propose a SPDK-based KVS that does not require a file system.
- SPDK is a user level device driver designed to provide high performance.
  - SPDK provides BDEV, a user-definable module.
  - BDEV can be inserted into the I/O path.
  - SPDK allocates and processes events to the core.
  - Each core is assigned an event loop, and the event loop consists of a reactor and an event queue.
  - Reactors handle events with threads in the core.
  - BDEV is inserted into the event queue in the form of an event and executed.

SPDK-based KVStore

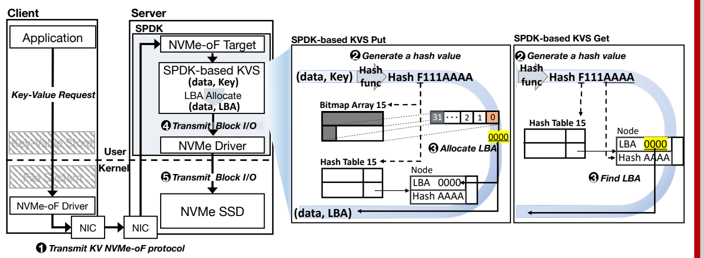

Figure 3. < I/O operation flow of SPDK based KVStore >

- SPDK-based KVS is implemented with two shared data structures.
  1) The hash data structure indexes a <key, value> tuple and manages the LBAs of the value of the key.
  2) Bitmap array manages free logical blocks on the storage.
- Processing steps for I/O operation flow

❶ A user-level application on the client send key-value requests to the server using the KV NVMe-oF protocol.

❷ In SPDK-based KVS, hash function generates a hash value of the received key.

❸ SPDK-based KVS operates for each type of KV request.
  - ***Put():*** An unused LBA is allocated by referring to the bitmap array, and the hash data structure is updated with the key and corresponding LBAs.
  - ***Get():*** Get the LBAs corresponding to the key in the hash data structure.

❹ The request is converted into Block I/O by obtaining LBAs and then transmitted to the user-level NVMe Driver.

❺ The NVMe driver notifies the client after writing the value to the NVMe SSD.

Evaluation

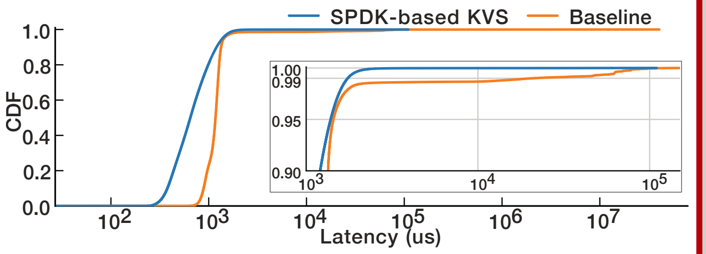

Figure 4. < Performance evaluation of SPDK-based KVS >

- We used two servers connected via 10 Gbps Ethernet, each server equipped with 10 cores and running Linux (Ubuntu 20.04).
- The storage server is equipped with Samsung 970 EVO 500 GB NVMeSSD.
- SPDK-based KVS was implemented using SPDK v.21.10.
- We used the write workload "Fillsequential" with a 32KB value of RocksDB's *db_bench* benchmark.
- We compared Figure 1(a) (Baseline) and Figure 1(d) (SPDK-based KVS).
  1) Baseline: RocksDB runs upon EXT4 on the client in Figure 1(a)
  2) SPDK-based KVS: our proposed approach of Figure 1(d)
- 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) (Development of low-latency storage module for I/O intensive edge data processing) under Grant 2020–0–00104.