

Transaction Support Using Compound Commands in Key-Value SSDs

Sang-Hoon Kim^{*}, Jinhong Kim[†], Kisik Jeong[†], and Jin-Soo Kim[‡]

HotStorage'19



^{*}



[†]



[‡]

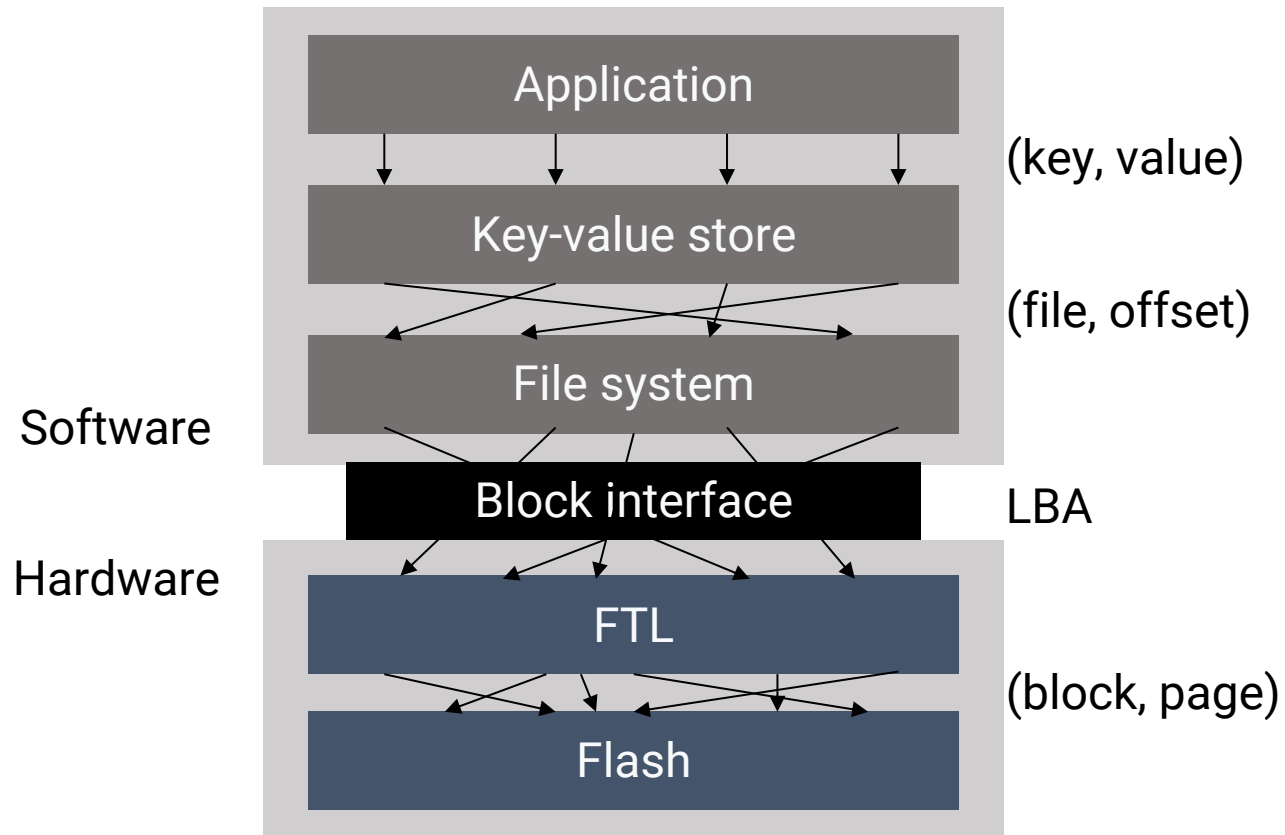
Key-Value Store is Popular and Important

- Simplify storing semi-structured data
- Accelerate enterprise services
- Also, as backend for distributed file systems
 - E.g., Ceph RADOS



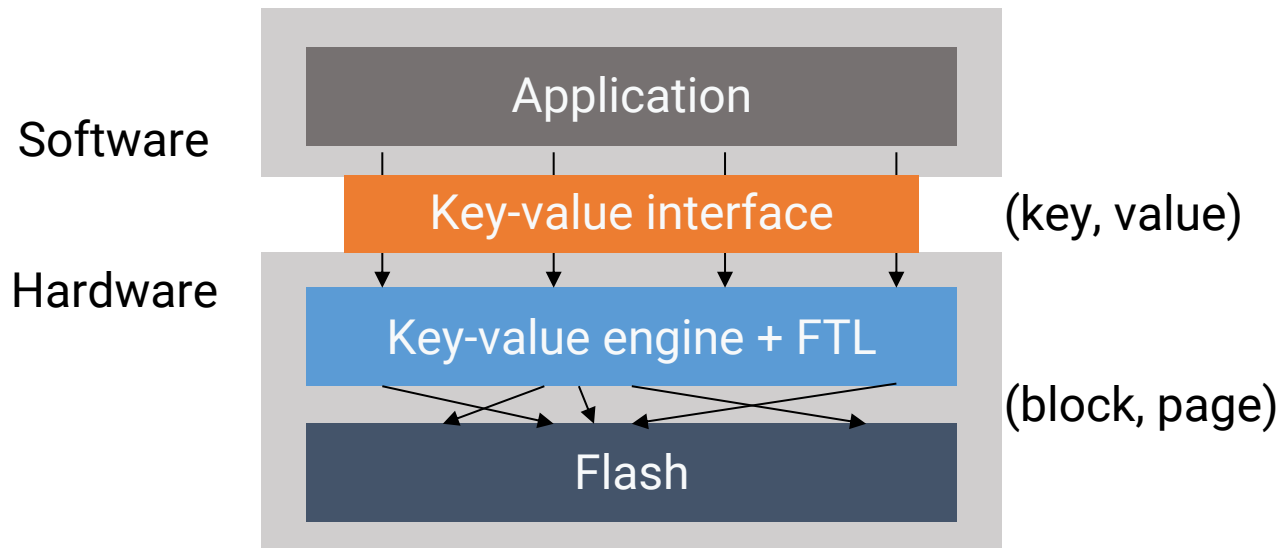
Downsides of Software Key-Value Store

- Deepen I/O stack, incurring additional overhead
 - Writes are *significantly* amplified (e.g., LSM tree)
 - Reads take long




Key-Value SSD (KVSSD)

- Provide the key-value store service at a device level
 - Allow to replace the deep software stack with off-the-shelf devices
- Promise performance improvement yet simplify storage management
 - Directly respond to data requests from an application with minimal involvement of the host software





Prototype KVSSD


Specification	
Interface	NVMe / U.2
Key size	4 – 255 bytes
Value size	0 byte – 2 MB
KV commands	Store, Retrieve, Delete, Exist, Iterate




Open Memory Platform Development Kit

 Repositories 5

 Packages

 People

 Projects

Find a repository...

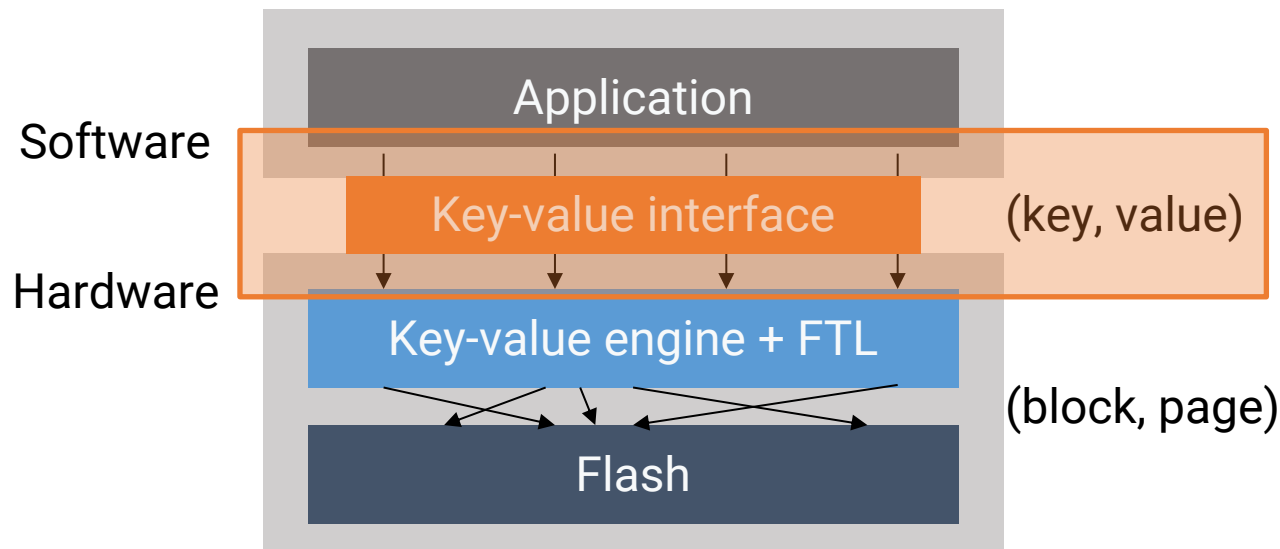
Type: All ▾

Language: All ▾



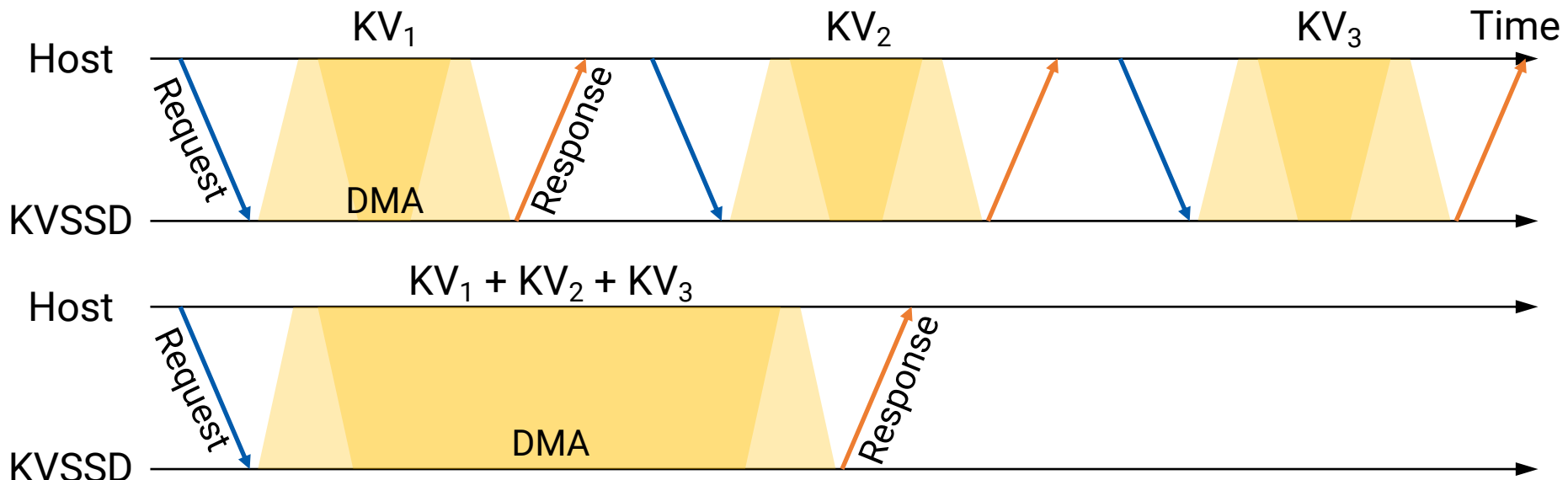
But, Current KV Interface is Too Simple

- Each KV operation is independently processed
- Thus, unable to group multiple operations together



But, Current KV Interface is Too Simple

- Each KV operation is independently processed
 - Thus, unable to group multiple operations together
- ➔ High interfacing overhead for small KV operations
- Common in KV store workloads



But, Current KV Interface is Too Simple

- **Each KV operation is independently processed**
- Thus, unable to group multiple operations together
- ➔ High interfacing overhead for small KV operations
 - Common in KV store workloads
- ➔ Unable to support transaction comprised of multiple KV operations

Compound Command

- Extend the original KVSSD command set to overcome the limitations
- Allow multiple KV pairs/operations to be packed in a single NVMe command
- And specify a semantic on the key-value group
 - E.g., transaction, locality group

List of Contents

- Introduction
- Extending KVSSD Command Set
- User API
- Transaction Support
- Evaluation

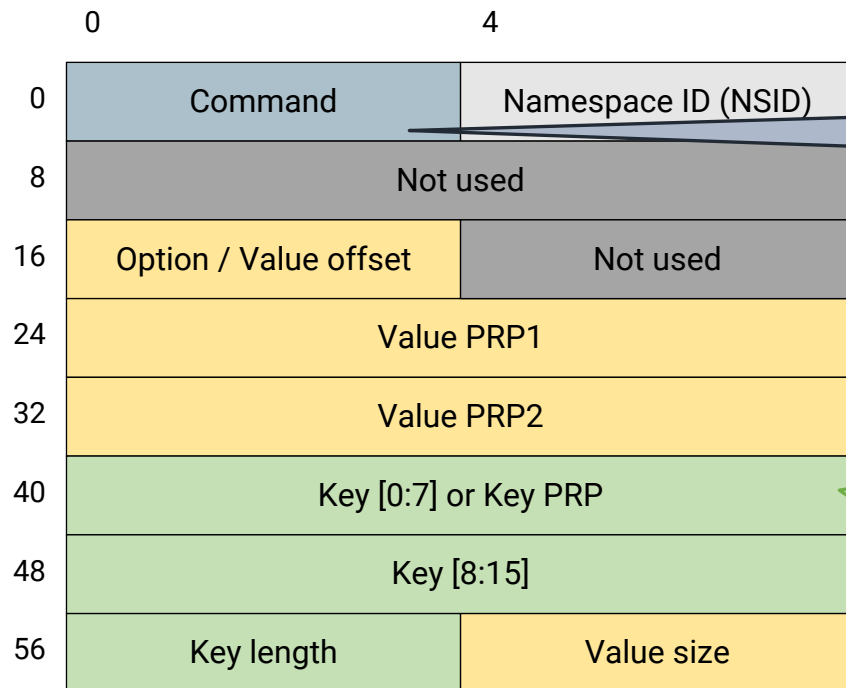
KVSSD Command Set

- The original NVMe command format
 - Each NVMe operation is translated to a 64-byte NVMe command

	0	4
0	Command	Namespace ID (NSID)
8	Not used	
16	Metadata Pointer (MPTR)	
24	Data Pointer (PRP) 1	
32	Data Pointer (PRP) 2	
40	Command specific DWORD 10/11	
48	Command specific DWORD 12/13	
56	Command specific DWORD 14/15	

KVSSD Command Set

- Extends the NVMe command set standard
 - Supports KV operations; Store, Retrieve, Delete, Exist, and Iterate

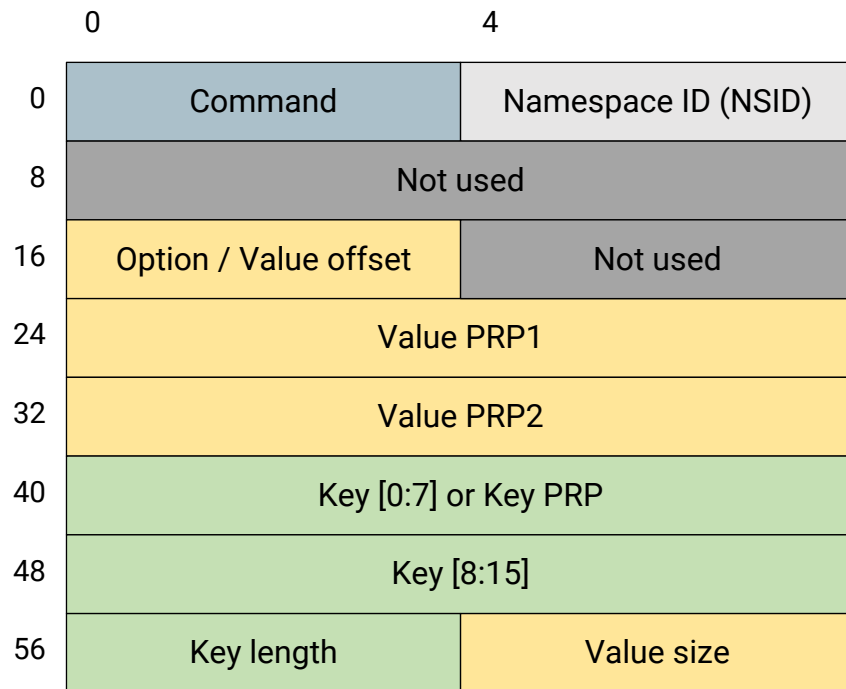


- Extended to include key-value operations

- Small keys (≤ 16 bytes) can be inlined in the command
- Otherwise, stored in memory and its location is specified
 - KVSSD fetches it through DMA

KVSSD Command Set

- Extends the NVMe command set standard
 - Supports KV operations; Store, Retrieve, Delete, Exist, and Iterate

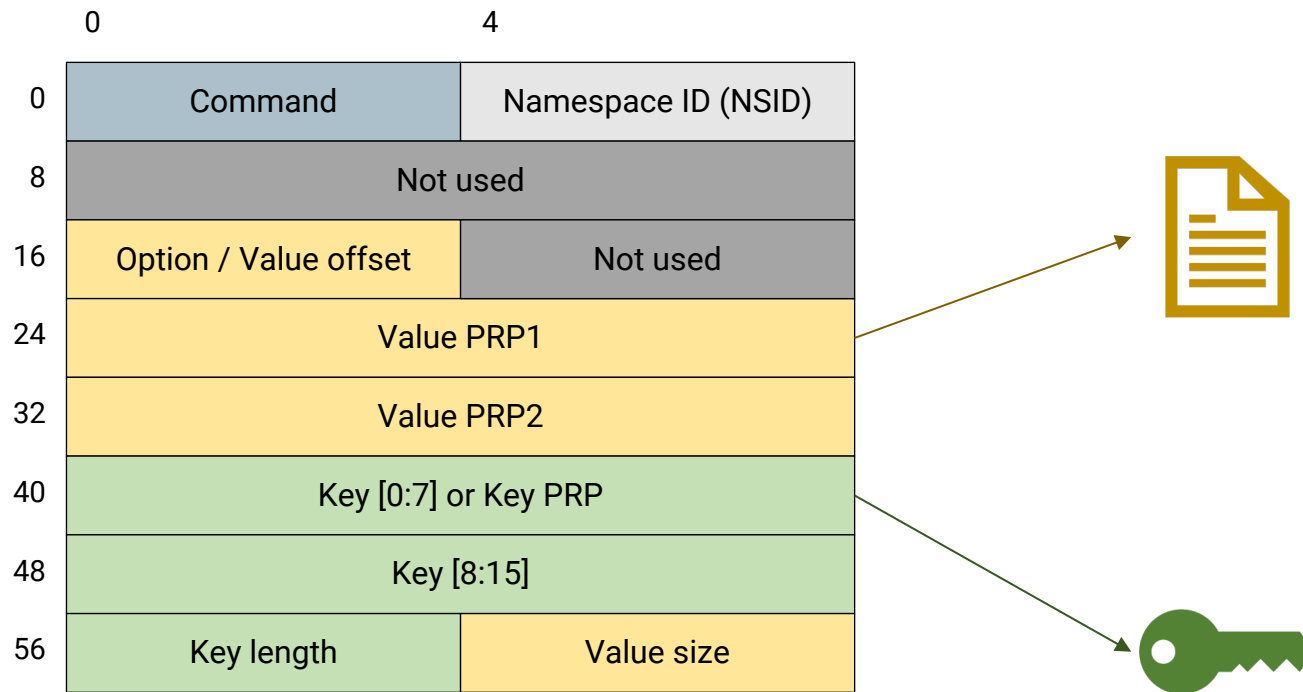


- Always stored in memory and its address is passed
- Long value can be composed with a scatter-gather list



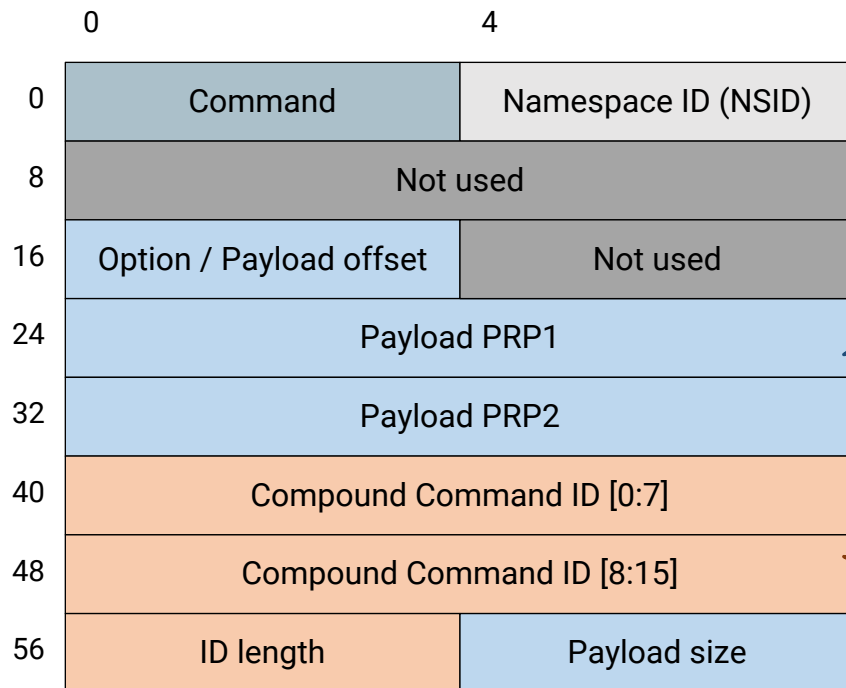
KVSSD Command Set

- Extends the NVMe command set standard
 - Supports KV operations; Store, Retrieve, Delete, Exist, and Iterate
 - KVSSD may need up to 2 DMAs to fetch key and value



Extending KVSSD Command Set

- Compound command uses the same command format
 - CC identification number + CC payload

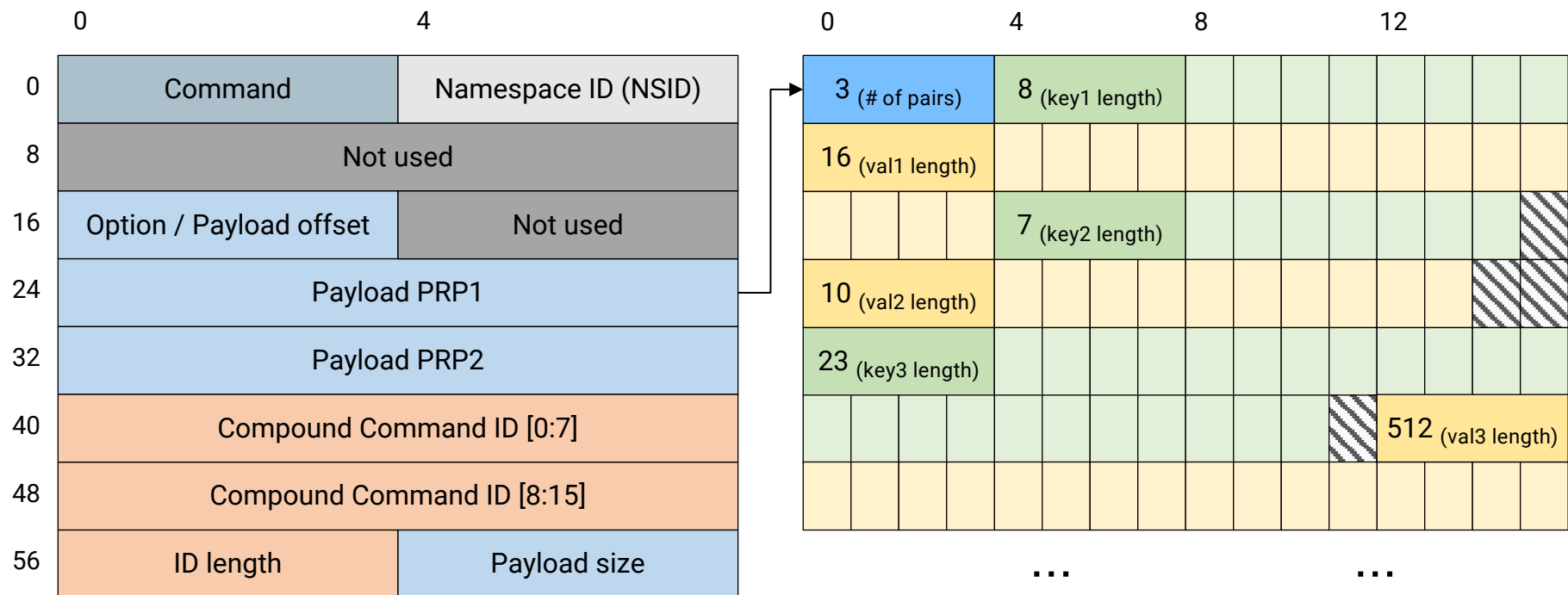


- Specify the location of payload
- KVSSD can fetch the entire payload with a single DMA operation

- 16-byte identification number
- Can be used as TX ID or group ID

Extending KVSSD Command Set

- Compound command uses the same command format
 - CC identification number + CC payload



User API for Compound Commands

```
int ret;  
CC_HANDLE *h;  
  
h = begin_compound(CC_GROUP);  
ret = retrieve(h, k1, k1_len, v1, &v1_len);  
ret = retrieve(h, k2, k2_len, v2, &v2_len);  
ret = retrieve(h, k3, k3_len, v3, &v3_len);  
ret = commit_compound(h);  
  
h = begin_compound(CC_TRANSACTION);  
ret = store(h, k1, k1_len, v1, v1_len);  
ret = store(h, k2, k2_len, v2, v2_len);  
ret = commit_compound(h);
```

Define a handle for CC

Start building CC

Append retrieve operations
for k1, k2, and k3

Submit the CC

Start building CC for TX

Append store operations
for k1 and k2

Submit the transaction

User API for Compound Commands

```
int ret;  
CC_HANDLE *h;
```

Define a handle for CC

```
h = begin_compound(CC_GROUP);  
ret = retrieve(h, k1, k1_len, v1, &v1_len);
```

Start building CC

Append retrieve operations

How can KVSSD internally process compound command TXs?

```
h = begin_compound(CC_TRANSACTION);  
ret = store(h, k1, k1_len, v1, v1_len);  
ret = store(h, k2, k2_len, v2, v2_len);  
ret = commit_compound(h);
```

Start building CC for TX

Append store operations
for k1 and k2

Submit the transaction

Transaction Support in KVSSD

- Option 1: Combine well-known policies and mechanisms
 - Disallow other operations to be interleaved between transaction operations (A, I)
 - Use WAL techniques to guarantee atomic update (A, D)
 - Safely and properly store data on the flash memory (D)
- Check consistency restrictions, and proceed only if operations do not violate the restrictions (C)
 - Allow simple implementation and API
 - But, only provide limited consistency
 - Unable to mix different operation types in a compound command

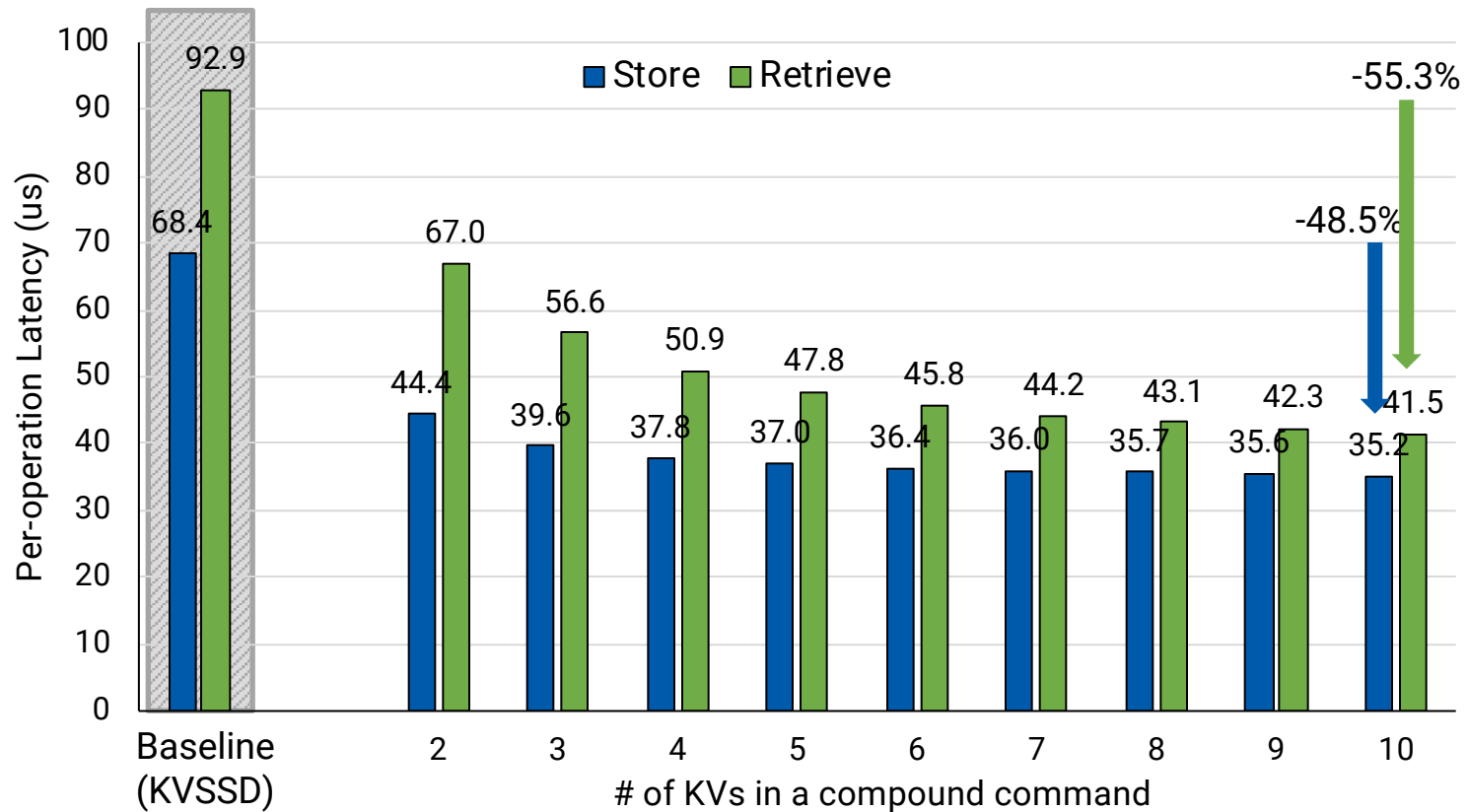
Transaction Support in KVSSD

- Option 2: Apply Amazon DynamoDB's approach
 - Application specifies preconditions for a transaction
 - E.g., Increase value of A by 10 only if A is less than 20
operation precondition
 - KVSSD processes KV operations while ensuring the preconditions
 - Can support sophisticated consistency models
 - Can process transactions concurrently, improving performance
 - But, require complicated APIs

Evaluation

- It would be best to modify the KVSSD firmware to make KVSSD understand the compound command and handle it accordingly
- However, it was not feasible at the time of paper submission
- Instead, estimate the performance
 - Measure the fundamental performance from the prototype KVSSD
 - Calculated the performance from measurement result
- Please see the paper for the detail

Evaluation: Microbenchmark

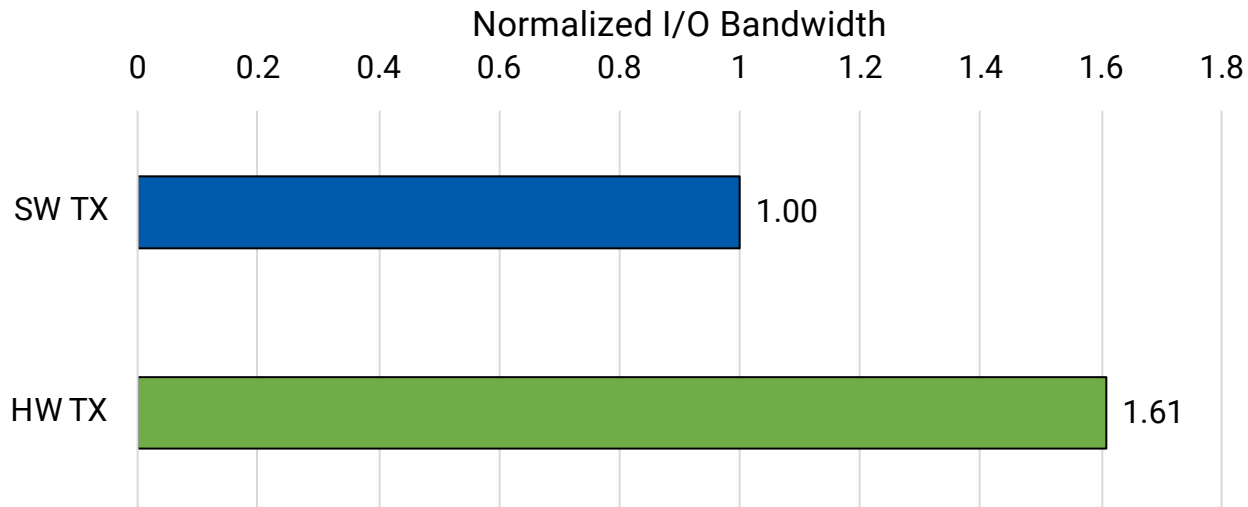


Evaluation: I/O Bandwidth

- We cannot evaluate from real devices
- Instead, we used KVEMU
 - Extension of FEMU (USENIX FAST'18)
 - Provides a virtual KVSSD device to guest OSes in virtualized environments
 - Emulate operation latency using latency models
 - Used the microbenchmark result for the latency model

Evaluation: I/O Bandwidth

- KVCeph on KVEMU
 - SW TX: Modified KVCeph to properly handle transactions using WAL-style update
 - HW TX: KVEMU processes compound commands by logging payload and then atomically switching index and data



Conclusion and Ongoing Work

- KVSSD promises to renovate the deep I/O stack
- But its KV interface is not properly designed to handle multiple small operations and support transactions
- Compound command complements the interface
- Performance optimization up to 55%
- To make KVSSD understand compound commands
- Working on making KVEMU public

Thank you!

Sang-Hoon Kim

sanghoonkim@ajou.ac.kr

<https://sslabslab.ajou.ac.kr>