Hybrid Data Reliability for Emerging Key-Value Storage Devices

Rekha Pitchumani

Yang-suk Kee

Memory Solutions Lab

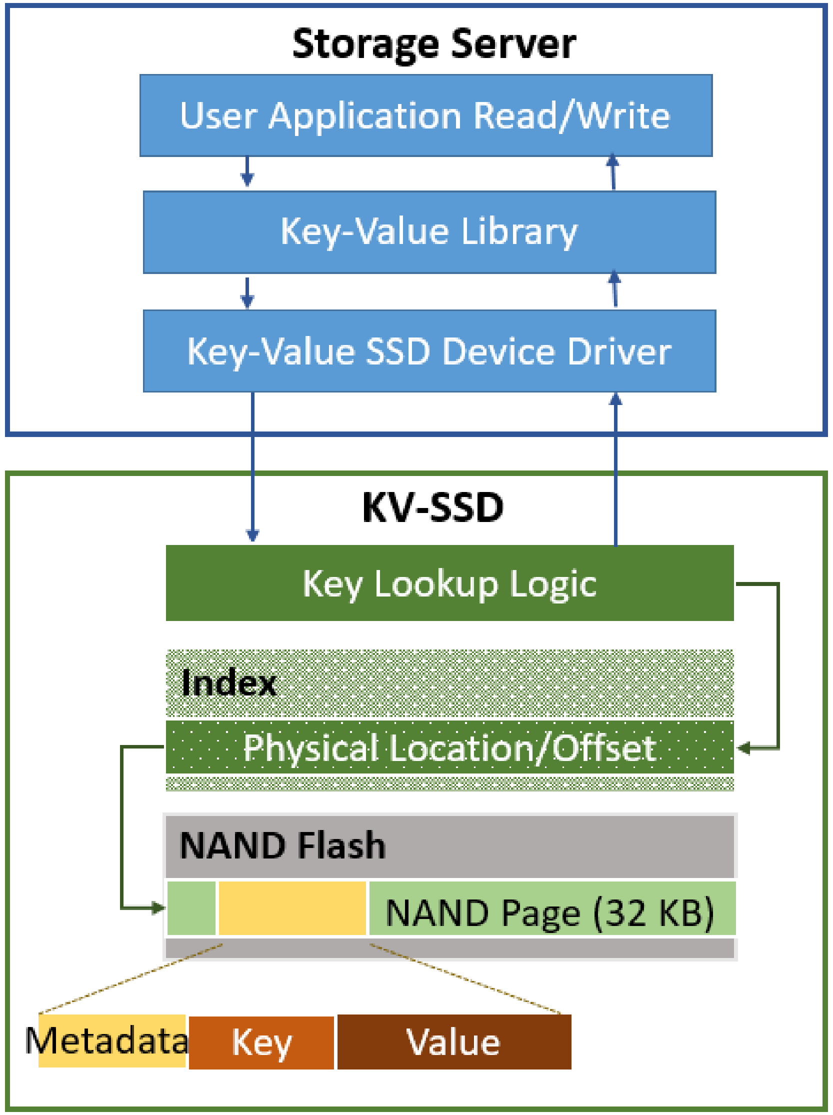
Samsung Semiconductor Inc

Summary

Emerging Key-Value Storage Devices Enable Providing Better Data Reliability (in many cases) at Competitive/Lower Cost on Throughput than Traditional RAID for Block Devices!!!

Key Value Storage Device

- Key-Value interface instead of traditional block interface
  - Store, retrieve and delete KVs
  - Check KV exist
  - Iterator support
- Thin host software stack
- SNIA standard Key Value Storage API Specification is available



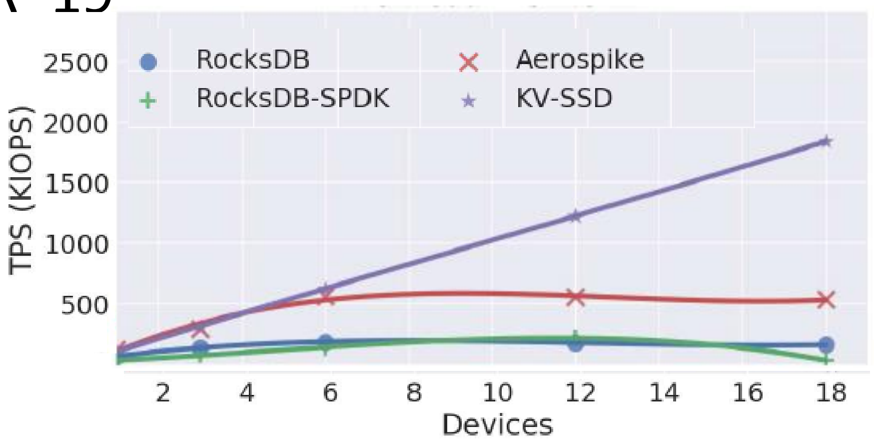
Prototype NVMe KV SSD from Samsung

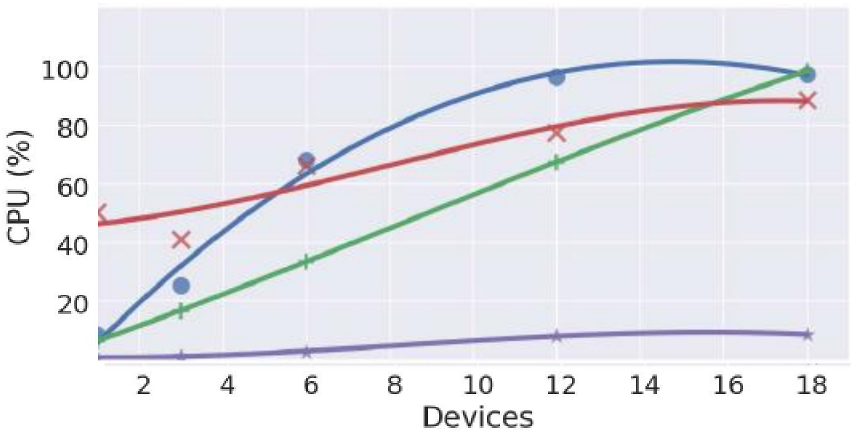
- **Same hardware as the enterprise-grade block SSD, but with KV firmware**
  - 4-255 byte keys and up-to 2 MB values

- For more on the ecosystem software, please check https://github.com/OpenMPDK

Details This Work Does NOT Go Into

- **KV IO throughput vs block IO throughput**
  - Depends on value size, key size; Prototype firmware
  - Not apples-to-apples - more internal tasks to do with same resources
- **How about more hardware resources for KV SSDs?**
  - Interesting question; Power, cost, etc.,
- **If KV SSD does not always beat block SSDs, why should I care?**
  - "Towards Building a High-Performance, Scale-In Key-Value Storage System". Kang et.al. SYSTOR '19
  - Little teaser →



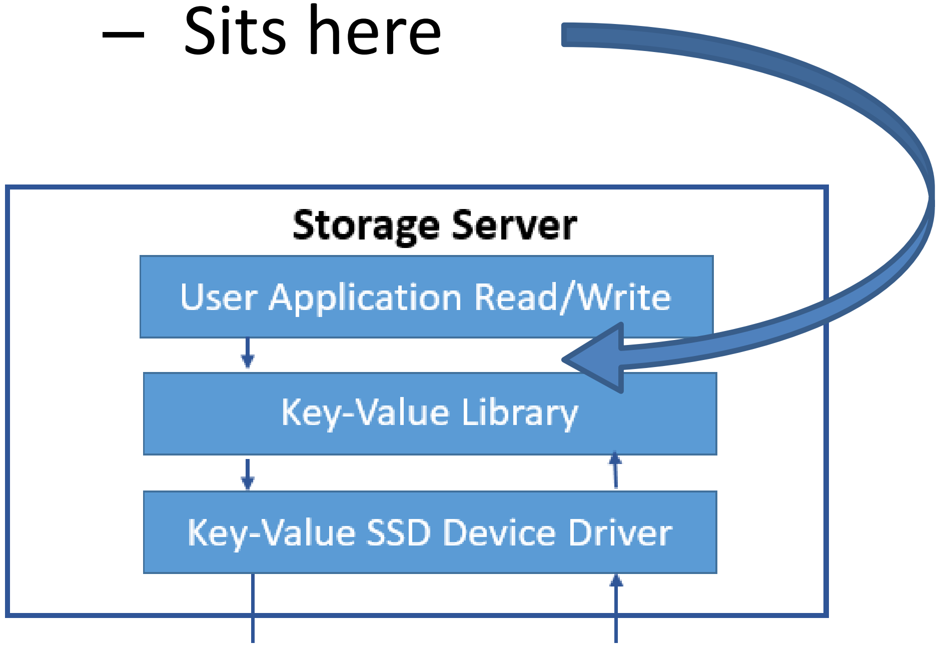


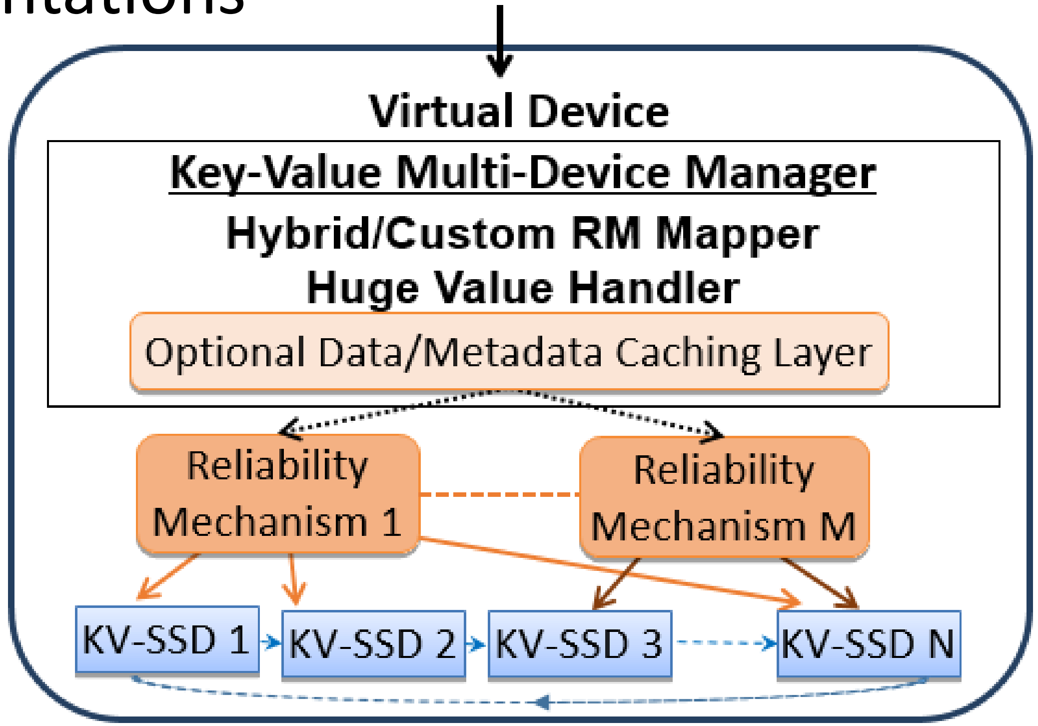
Data Reliability Requirements

- Multiple options with different trade-offs
  - Kind of like RAID for block storage devices
  - Suitable for variable-length keys and variable-length values
  - Should preserve the low host resource requirements of KV devices
- Flexibility and co-existence of multiple options

Key-Value Multi-Device (KVMD)

- **Hybrid data reliability manager for KV SSDs**
  - Stateless design
  - Multiple pluggable reliability mechanisms suitable for variable-length keys and values
  - Pluggable erasure code implementations
  - Sits here





Reliability Mechanisms (RM)

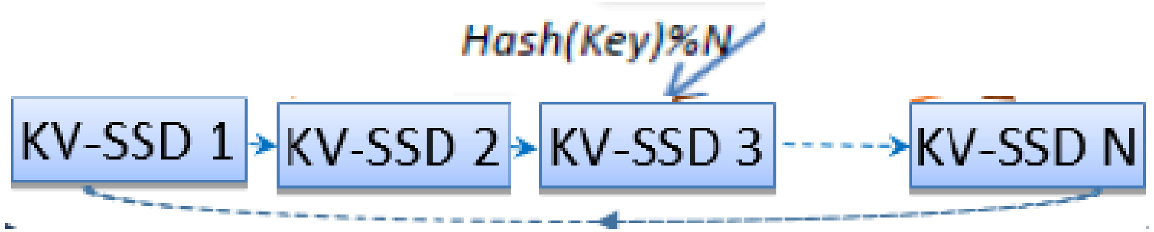
- Serves as counterparts to the traditional RAID0, RAID1, and RAID6 architectures
  - Hashing
    - No redundancy
  - Replication
    - Replicas
  - Splitting
    - Erasure Coding
  - Packing
    - Erasure Coding

Modes of Operation

- **Standalone mode**
  - Choose a single RM for all the KV pairs
- **Hybrid mode**
  - Configuration file based **–** different RMs for KVs in different value size ranges that co-exist
- **Custom mode**
  - Set either the *standalone* mode or the *hybrid* mode, and specify a RM per write
  - To be used for known outliers

Hybrid Mode

- **To co-exist in the hybrid mode, the RMs have to**
  - Place the first copy/chunk of the KV pair on the **primary device**, determined using the **same hash function** on the key, modulo the number of devices
  - Store at-least the first copy/chunk/info using the **same key** as the user key



Hybrid Mode & KVMD Metadata

- Store required metadata in the **beginning** of the value

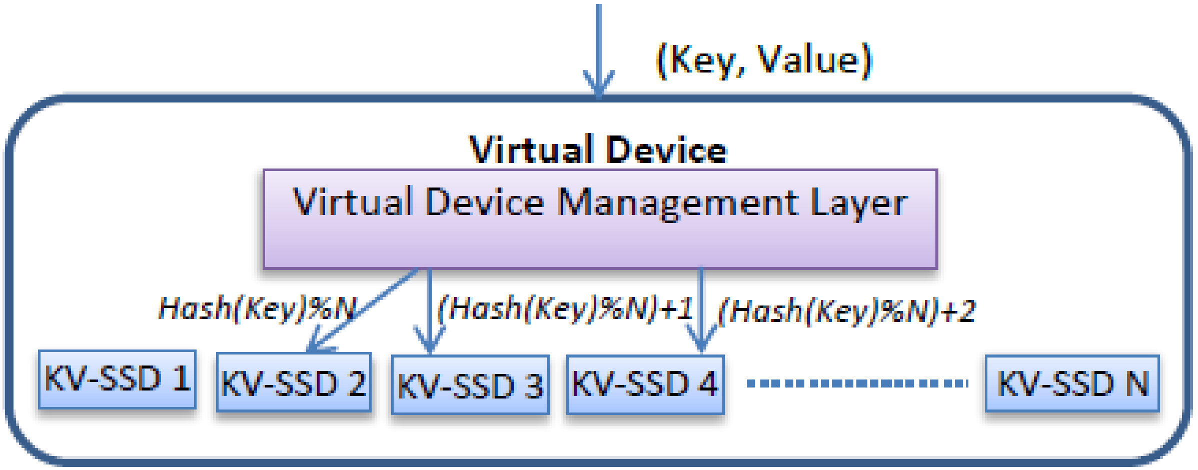
RM ID (1 byte)	EC ID (1 byte)	Total splits (2 bytes)	Checksum (4 bytes)	RM specific & Padding (8 bytes)	Full/Partial User Value (Variable Length)

- Hybrid Mode reads before any operation
  - Optional caching layer can help
- Huge Object handling

User Key (Variable length)	Split# (1byte)	RM-META (1byte)

Hashing & Replication

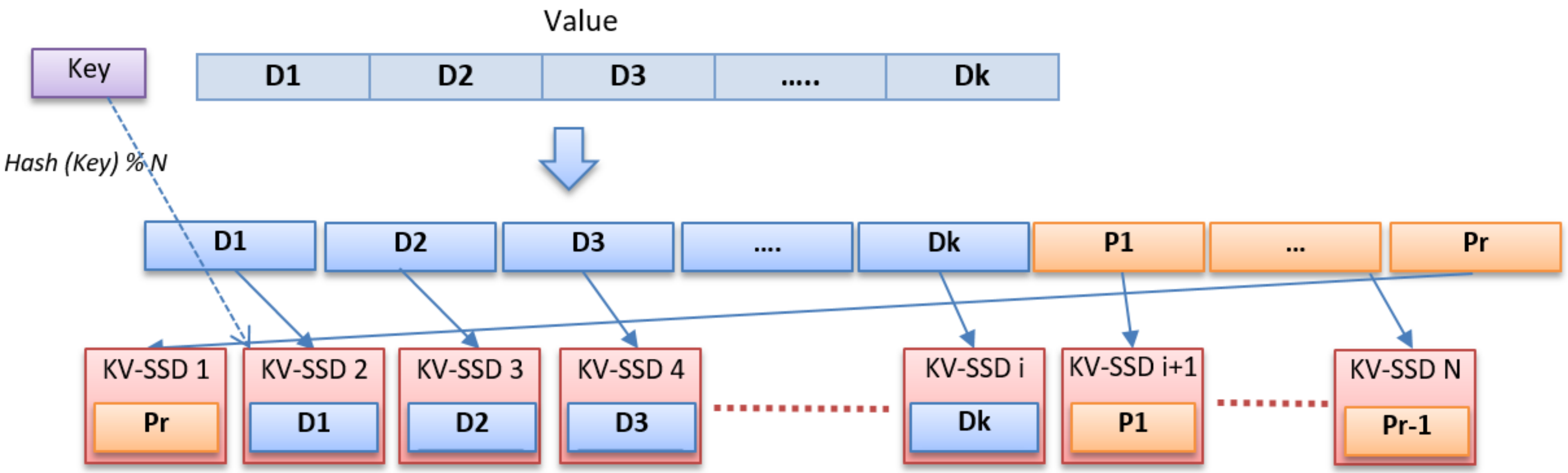
- Hashing
  - Similar to RAID0; Distributes KV objects.
- Replication
  - Similar to RAID1; Replicates KV objects



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1
RM	**Value Metadata**																**Key Metadata**
Hashing	1	0	Splits		Checksum				Padding								None
Replication	2	r	Splits		Checksum				Padding								None

Splitting

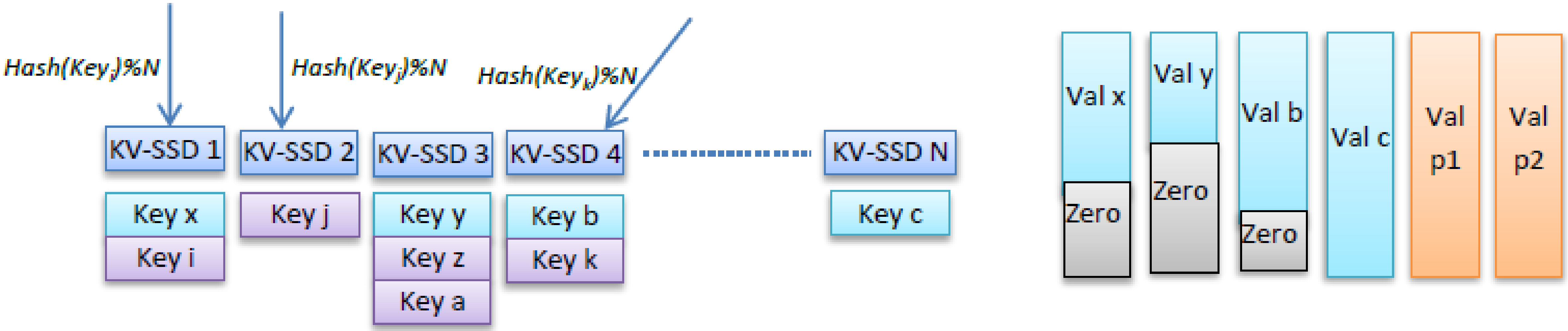
- Splits the value into k equal-sized objects and add r parity objects
- Configurable erasure coding methods



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1	···
RM	**Value Metadata**																**Key Metadata**	
Splitting	3	ec	Splits		Checksum				Value Size				k	r	Padding		None	

Packing

- Groups multiple KV objects
- Packs up-to k different objects into a single reliability set
- Configurable erasure coding methods & virtual zero padding

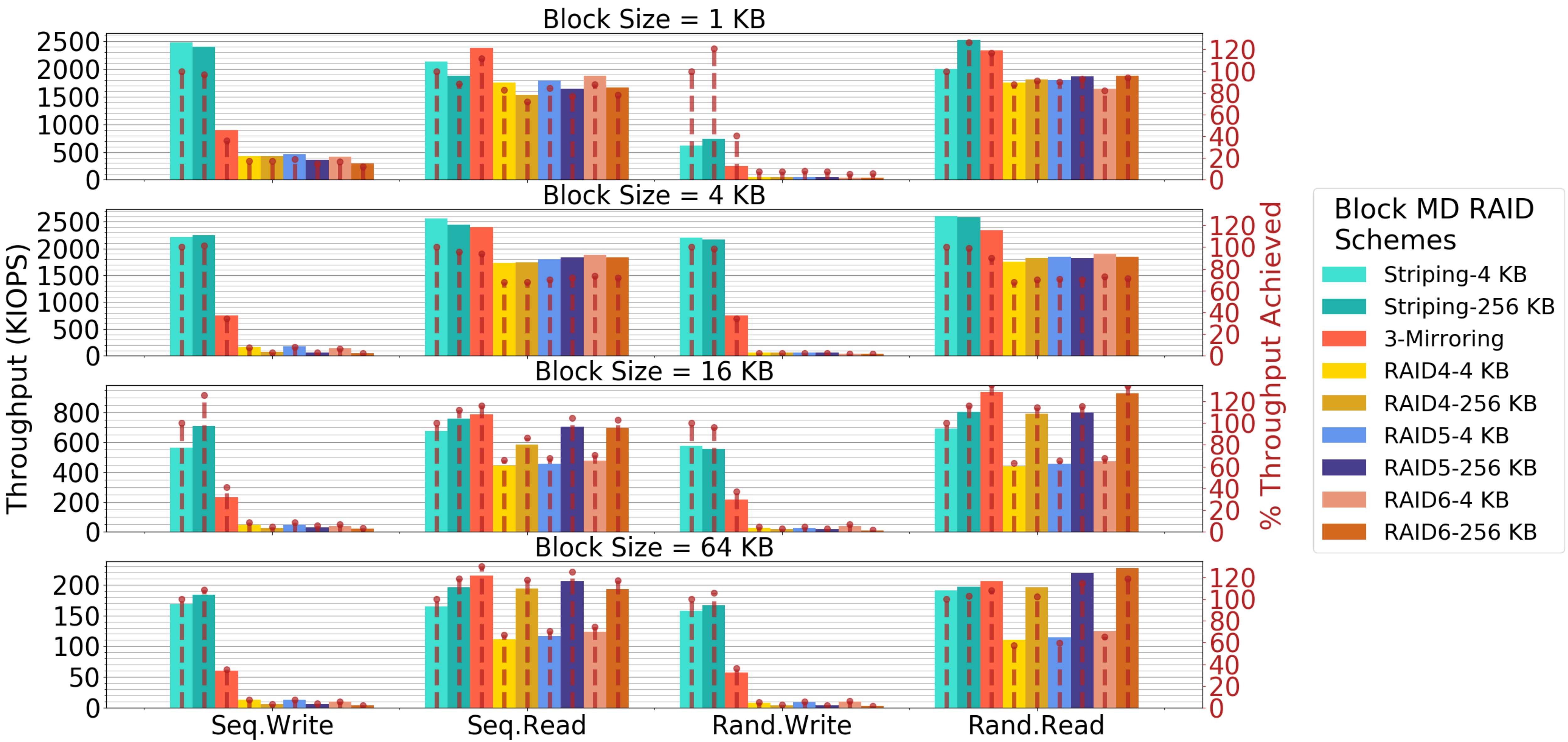


RM	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1	⋯
	Value Metadata																**Key Metadata**	
Packing	4	ec	Splits		Checksum				k	r	Padding						U/M/P	
	Metadata Value																	
Packing	ck	r	Key Size		Var-length Key				Value Size				Repeat $\cdots (k+r-1$ more KVs)					

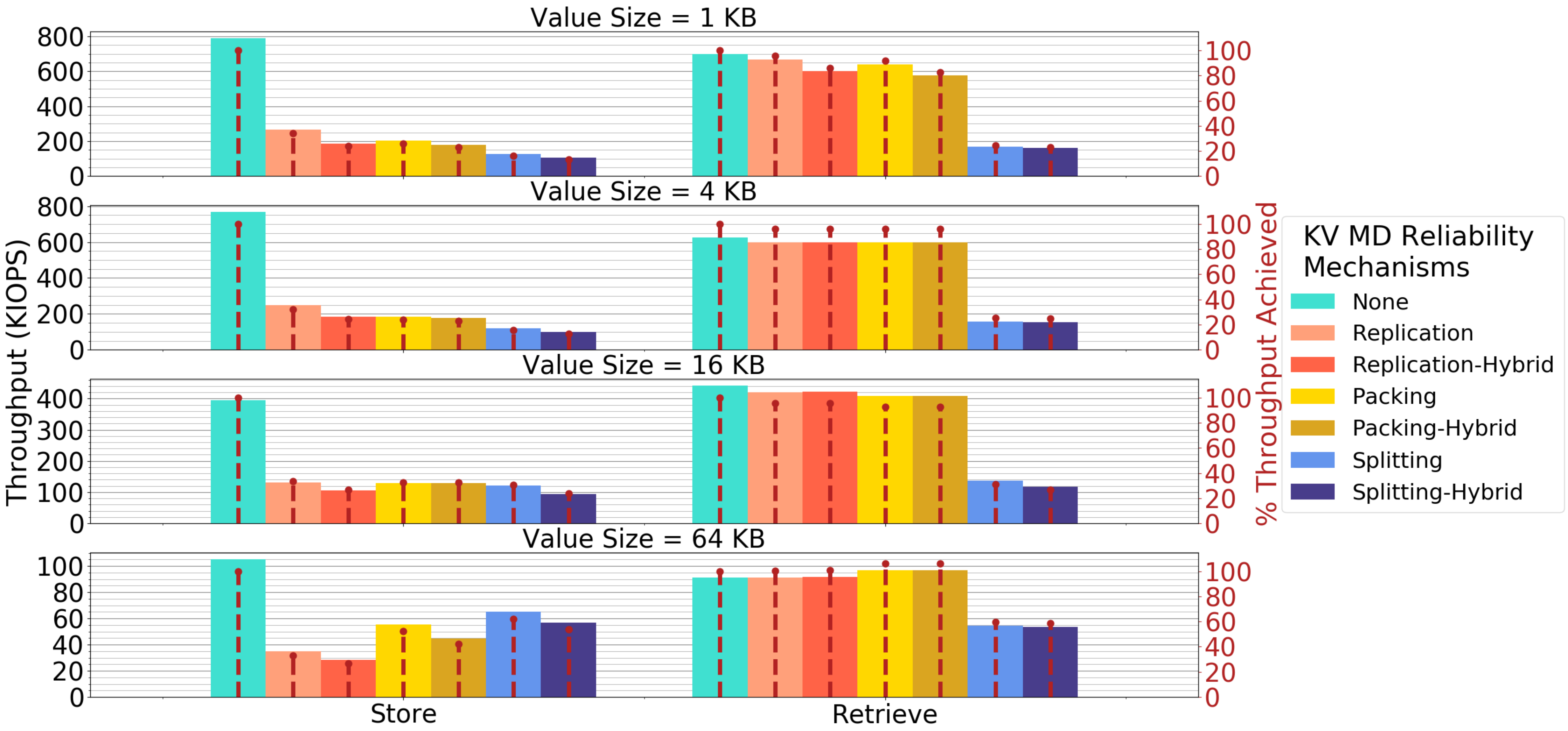
Evaluation

- **Evaluate software RAID (Linux Mdadm) for block devices and KVMD reliability mechanisms for KV SSDs**
  - Used the same 6 NVMe SSDs with different firm wares
- **KVMD also has hash calculations and 32-bit checksum calculation and verification overhead for every operation**
  - crc32 IEEE checksum calculation function using ISA-L library
  - Reed Solomon erasure coding implementation for any k and r using the ISA-L library

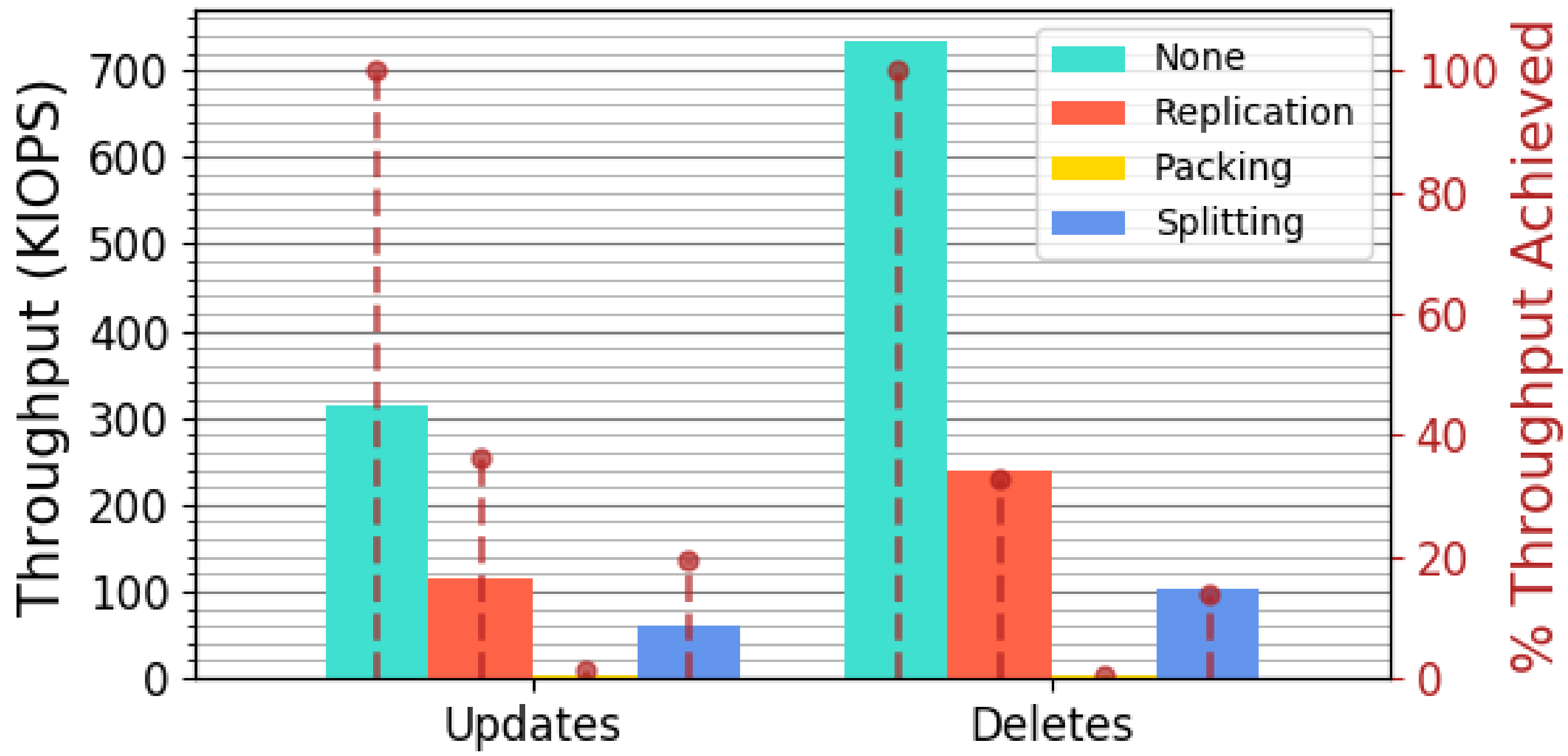
RAID's Cost on Throughput



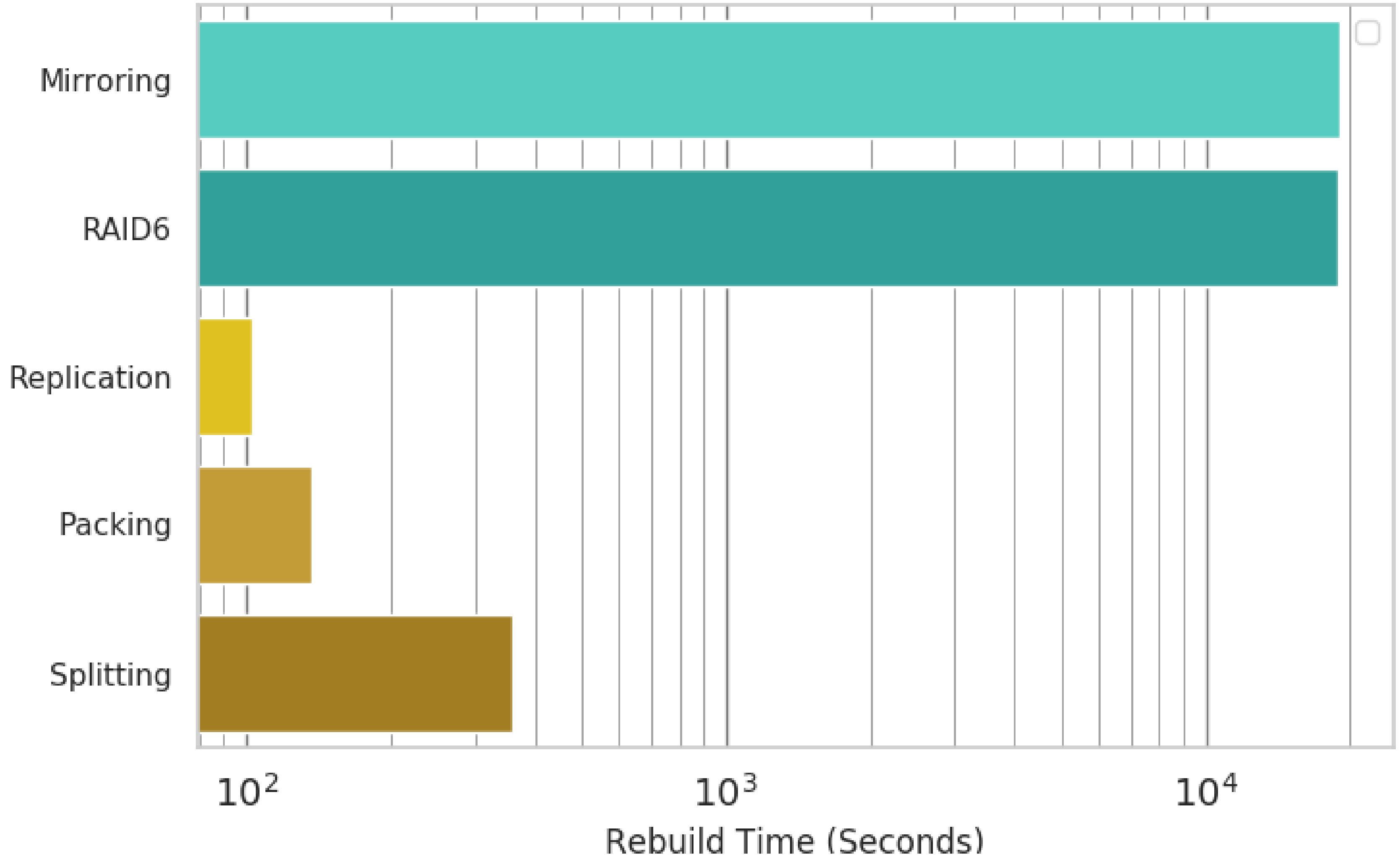
KVMD's Cost on Throughput



Updates and Deletes



Single Device Failure Rebuild



Limitations & Future Directions

- Data/Metadata Caching
- Versioned Updates
  - Packing performance
  - Concurrency control
  - Crash consistency
- Automatic RM Determination
- Even Capacity Utilization

Conclusion

- Better MTTDL than block SSDs in many cases due to
  - Reduced write amplification (Not yet for Packing updates)
  - Faster device rebuilds, proportional to number of user objects, rather than device capacity
- KVMD throughput degradation comparable to or lower than software RAID incurred degradation in most cases
- Offers high flexibility

Backup

Comparison

	Block SSD		KV SSD		
	RAID 1	RAID 6	Replication	Packing	Splitting
Writes	$1/r$	$[1/N, (N-2)/N]$	$1/r$	$[1/(N+m),$ $k/(N+m)]$ where m (metadata) $= [r, rk)]$	$1/N$
Reads	1	1	1	1	$[1/k, 1]$
Rebuild Time	$\Uparrow\Uparrow$ ($\propto$ Device capacity)	$\Uparrow\Uparrow$ ($\propto$ Device capacity	$\downarrow$ ($\propto$ Number of user objects)	$\uparrow$ ($\propto$ Number of user objects)	$\Uparrow$ ($\propto$ Number of user objects)
Space Utilization	$1/r$	$(N-2)/N$	$1/r$	$[1/(r+1), k/N]$ metadata is additional, but assumed small	k/N
Write Amplification	$\Uparrow$	[$\uparrow$ for stripe aligned and sized writes, $\Uparrow\Uparrow$ for most writes]	$\Uparrow$	$\uparrow$ for inserts $\Uparrow\Uparrow$ for updates	$\uparrow$
Pros & Cons	Similar writes for all sizes. Best reads. Low MTTDL due to WA.	Very poor writes and good reads. Poor, workload-dependent MTTDL due to WA.	Similar to RAID 1. Best for small, hot objects.	Best reads. Best inserts. Very poor updates. Good, workload-dependent MTTDL.	Writes/reads $\propto$ value & request sizes. Best MTTDL. Best for large values.