DIDACache: A Deep Integration of Device and Application for Flash based Key-value Caching

Zhaoyan Shen†, Feng Chen‡, Yichen Jia‡, Zili Shao†

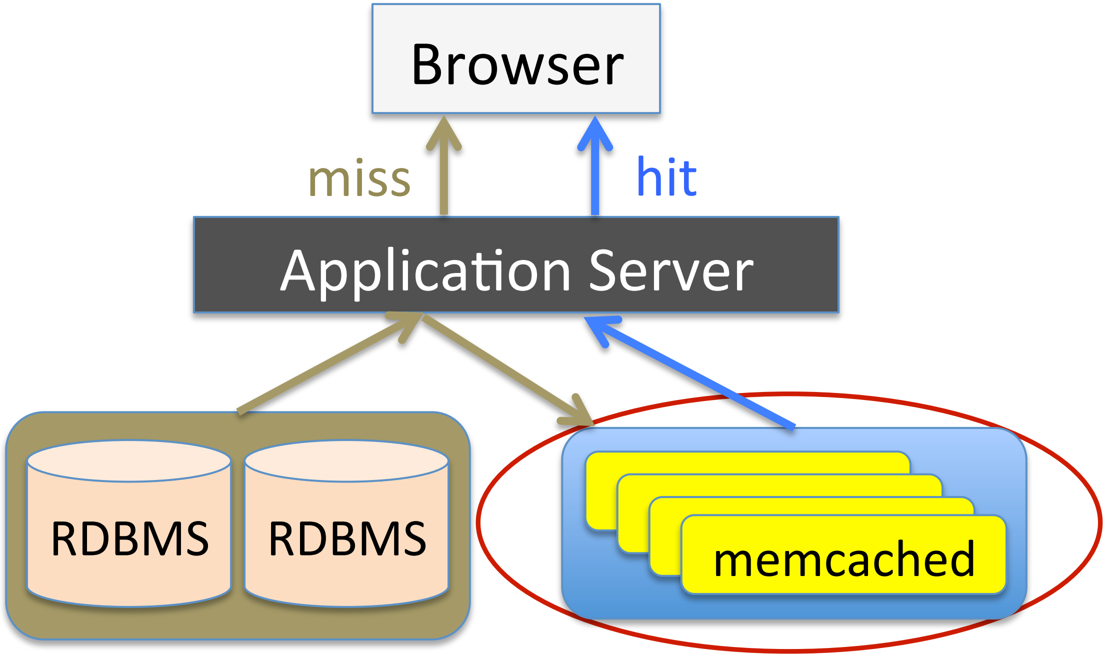
†Department of Computing, Hong Kong Polytechnic University

‡Computer Science & Engineering, Louisiana State University

The 15th USENIX Conference on File and Storage Technologies (FAST'17)

Key-value Information

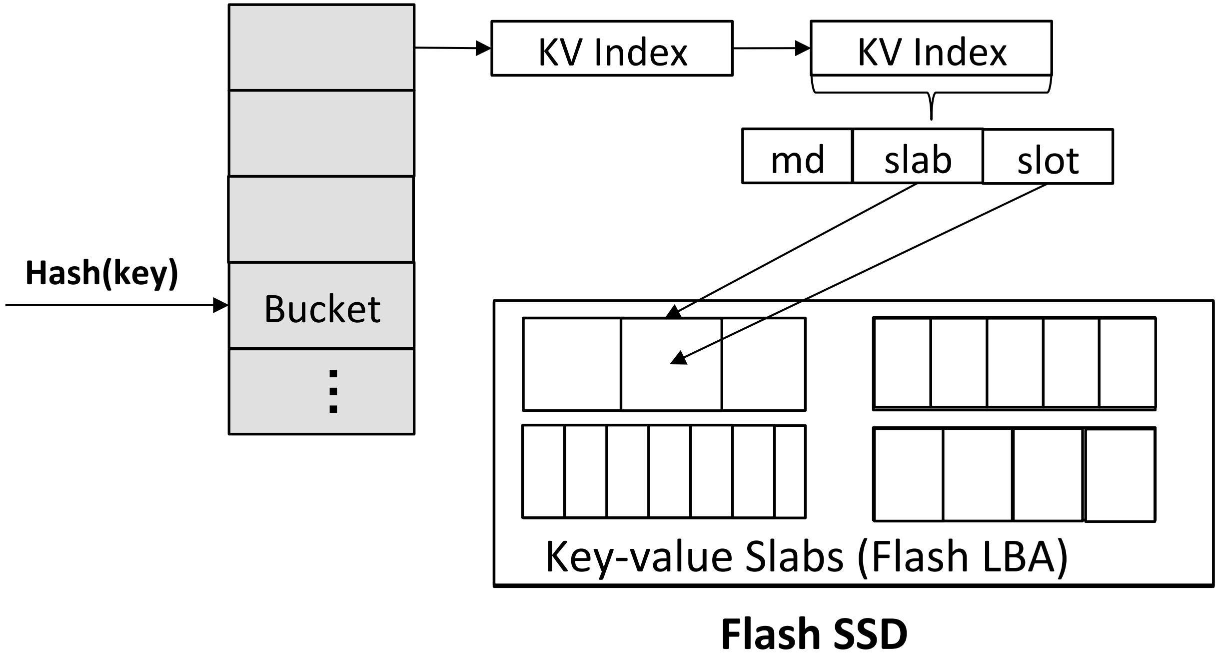
- Key-value cache is the first line of defense
  - Benefits: improve throughput, reduce latency, reduce server load
- Flash based key-value cache: McDipper, Fatcache



- In-memory KV cache
  - High access speed
  - High power consumption
  - High monetary cost
  - Capacity limitation

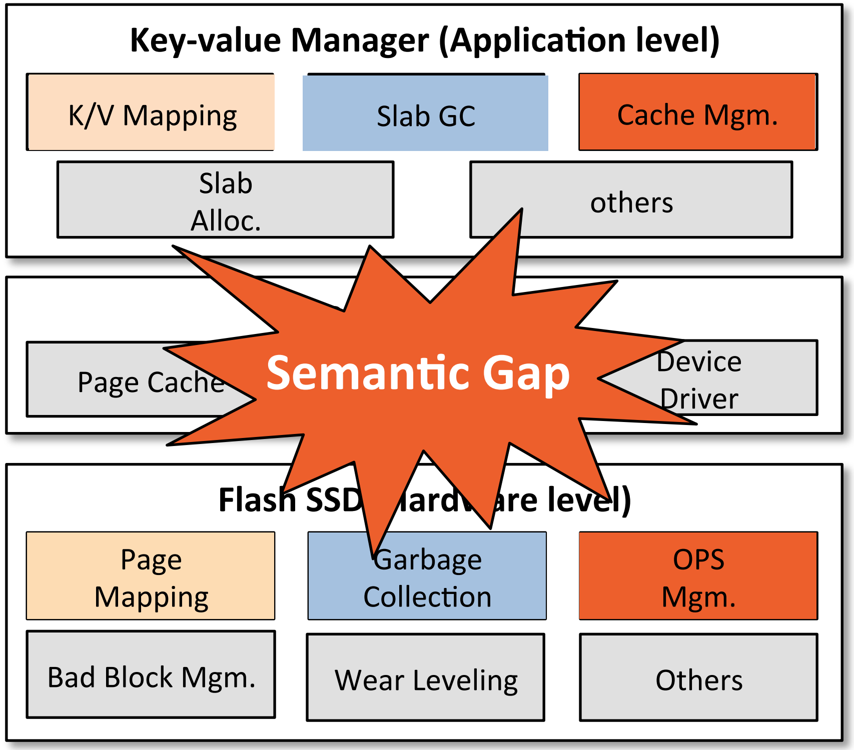
Flash based Key-value Cache

- Current Practice: Directly use commercial SSD as caching media



- In-memory hash table
- Log-structured slabs
- Out-of-place update

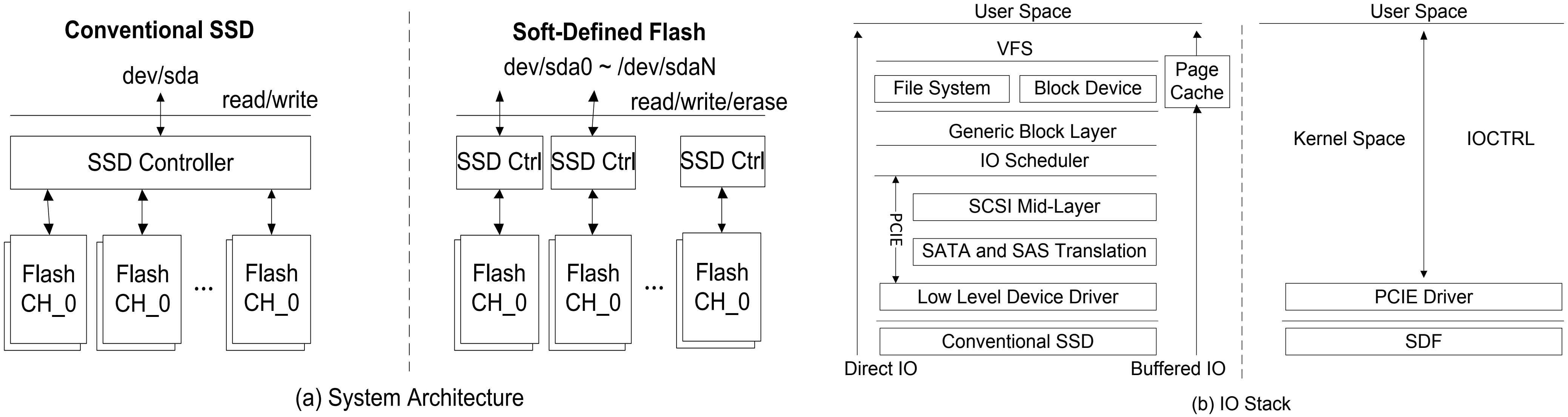
Research Issues



- Application level
  - Key-value mapping: key→slab
  - Slab-level GC (item granularity)
  - Cache management
- Hardware level
  - Page mapping
  - Flash page level GC
  - OPS management

Open-channel SSD

- Architecture & IO Stack



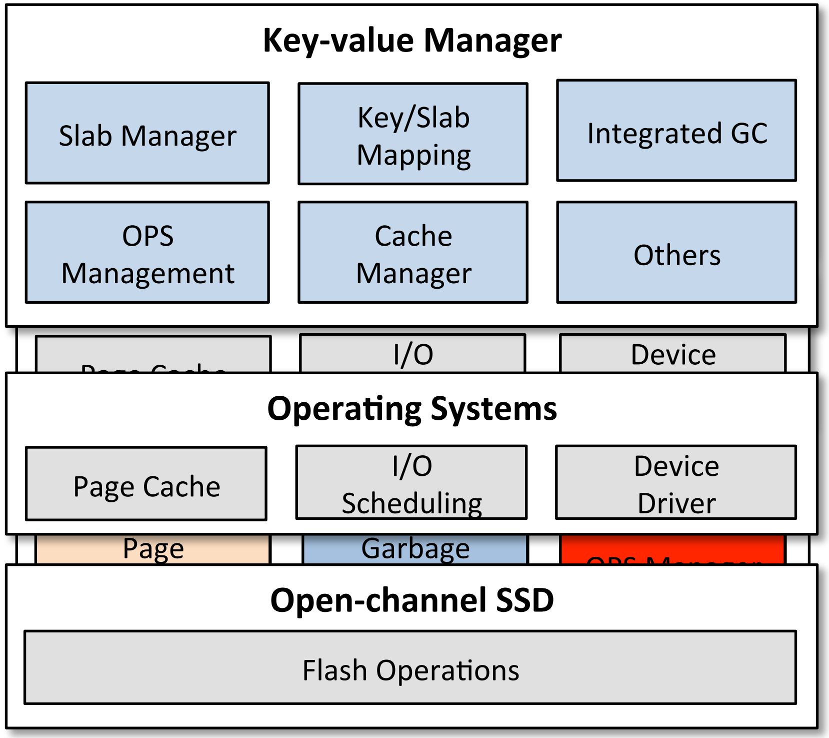
Open-channel SSD provides us unprecedented new opportunities.

[1] Ouyang Jian, et al., SDF: Software-Defined Flash for Web-Scale Internet Storage Systems (ASPLOS'14)

Related Work

- Flash-based key-value cache
  - Fusion-IO and Memcached： more cache, less headache
  - SSD-assisted hybrid memory to accelerate Memcached over high performance networks (ICPP'12)
- Open-channel SSD
  - SDF: Software-Defined Flash for Web-Scale Internet Storage Systems (ASPLOS'14)
  - LOCS: An Efficient Design and Implementation of LSM-Tree based Key-Value Store on Open-Channel SSD (Eurosys'14)
  - NVMKV: A Scalable, Lightweight, FTL-aware Key-Value Store (ATC'15)
- Key-value caching + Open-channel SSD
  - Optimizing flash-based key-value cache systems (Hotstorage'16)

DIDACache: An Enhanced Flash-aware Key-value Cache

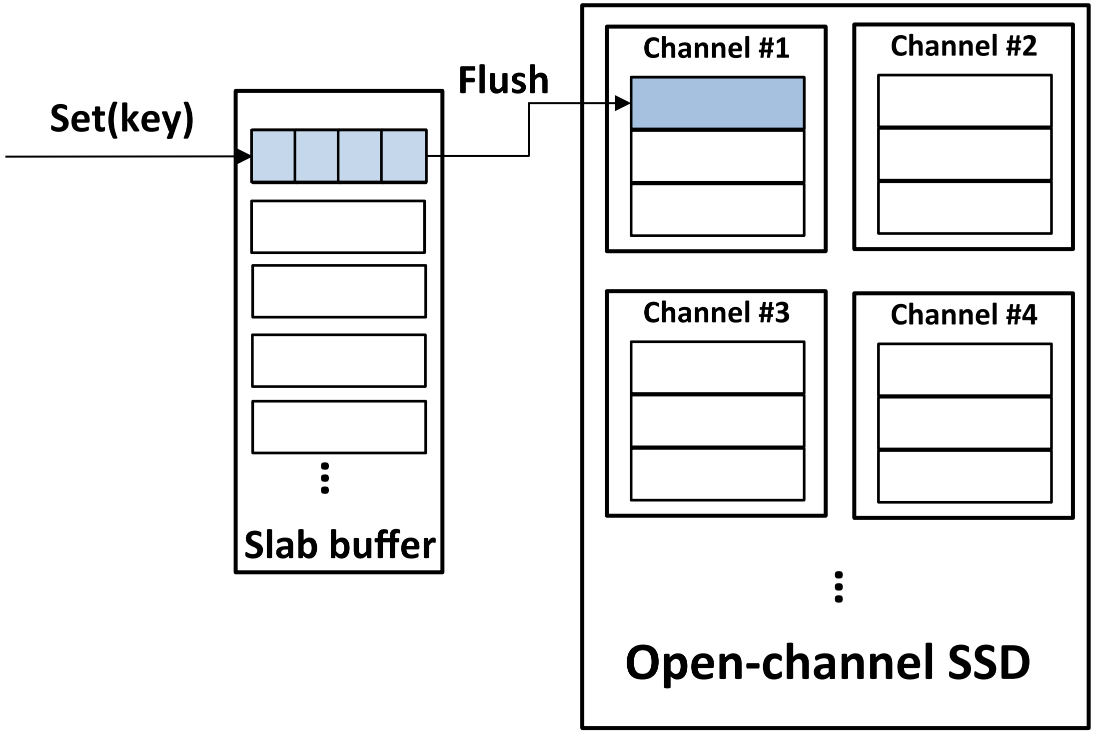


- Direct application driven
  → Fully exploit application semantics
- Hardware design simplified
  → Non-essential components removed
- Semantic gap issue mitigated
  → A tight application-device connection

DIDACache: An Enhanced Flash-aware Key-value Cache

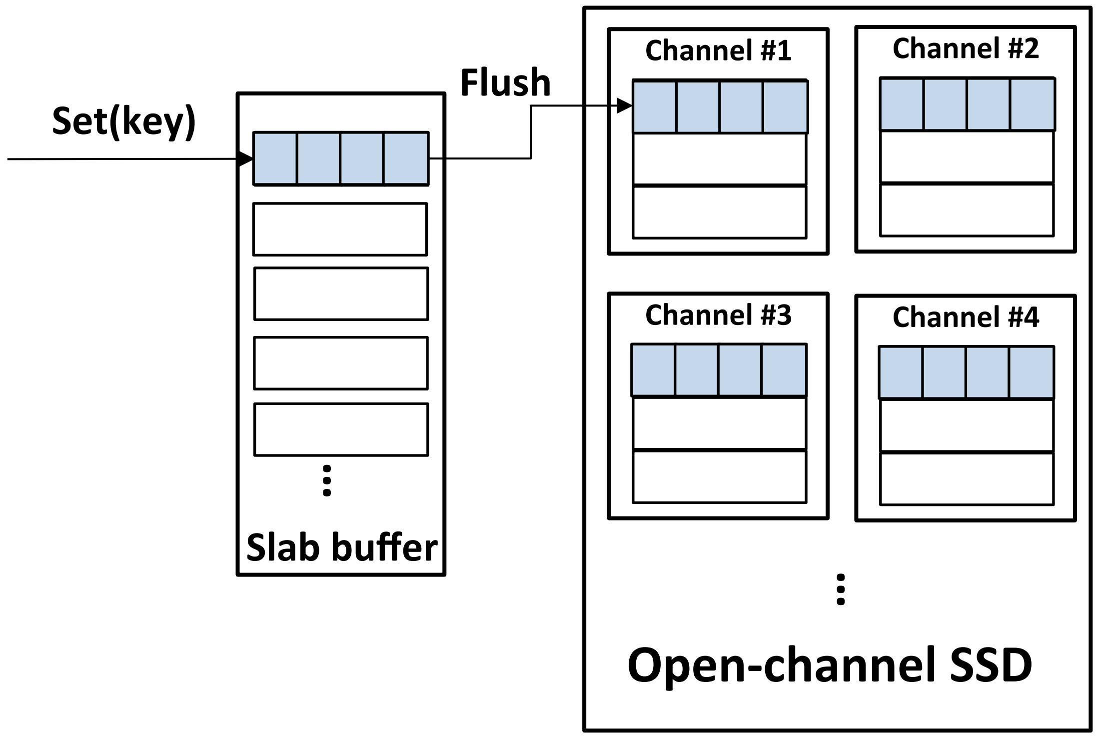
- Slab management
- Unified direct mapping
- Garbage collection
- OPS management

Slab Management: Slab buffer



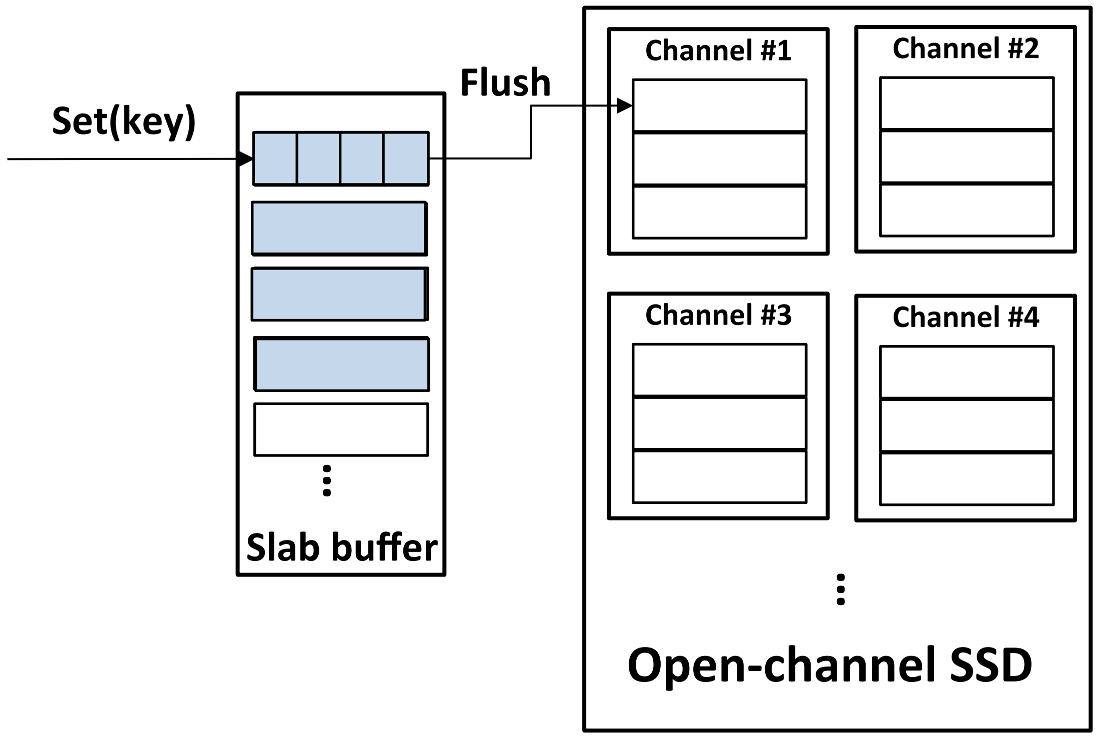
- Merge small requests
  → Organize big log-like writes
- Asynchronized requests
  → Hide I/Os from critical path
- Improve access speed
  → Immediate return

Slab Management: Slab-to-Channel Mapping



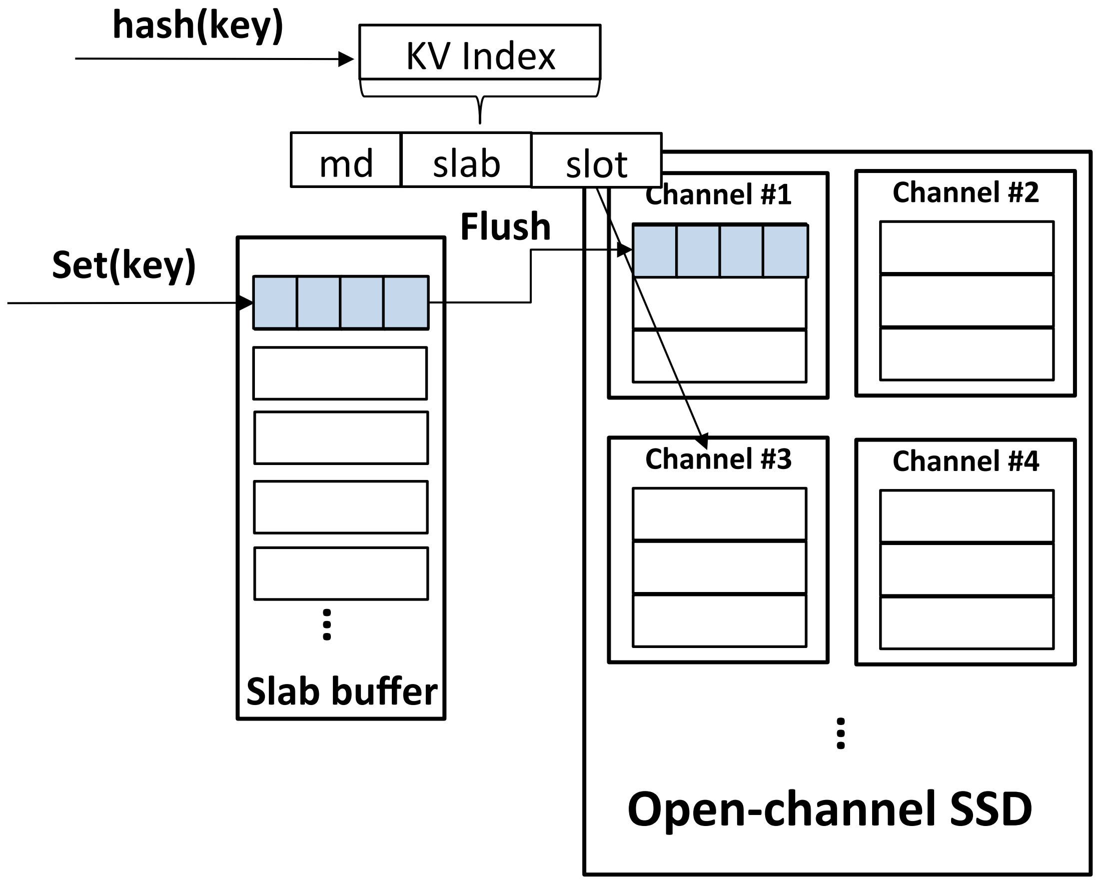
- Cross-channel mapping:
  - Slab sliced to chunks
  - Stripe chunks to channels
- Advantage:
  - Internal parallelism utilized
- Disadvantages
  - Complex mapping/space management
  - Small chunks → Sub-block writing/GC
  - Large chunks → Bad block, too big slab

Slab Management: Slab-to-Channel Mapping



- Per-channel mapping:
  - Slab size equals to one flash block
  - Static map a slab to one block
- Advantage:
  - No need of mapping structure
  - Transfer is efficient

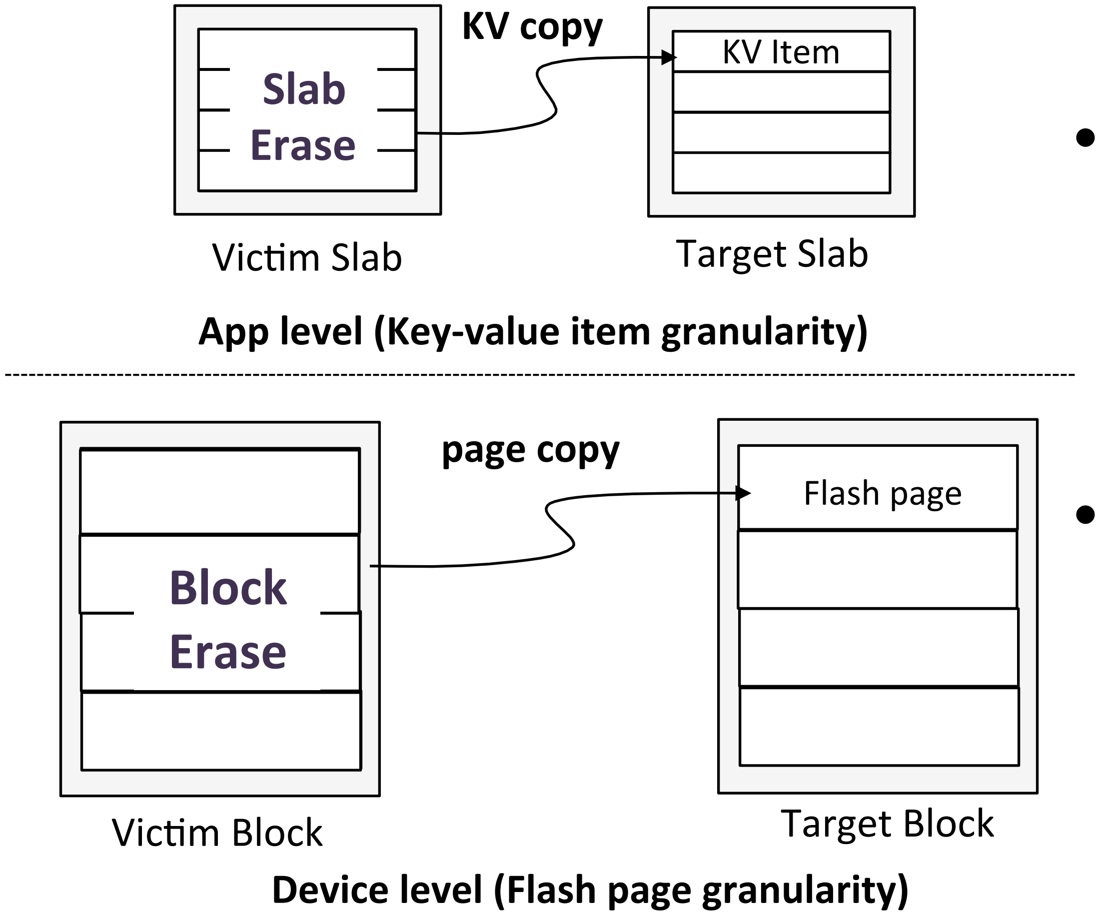
Slab Management: Simplified Mapping



- Unified mapping structure:
  - Direct key-to flash mapping
- Advantages:
  - Eliminate intermediate layer
  - Reduce DRAM consumption

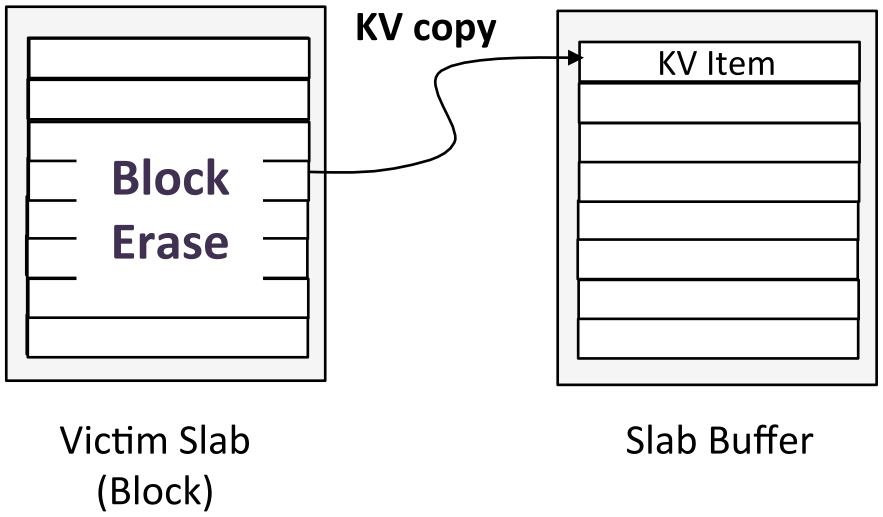
Integrated Garbage Collection

- Double garbage collection problem



- Double GC processes at two levels
  - Run simultaneously and independently
  - Run with different granularity
- Problems of double GC
  - No coordination
  - Redundant data copy

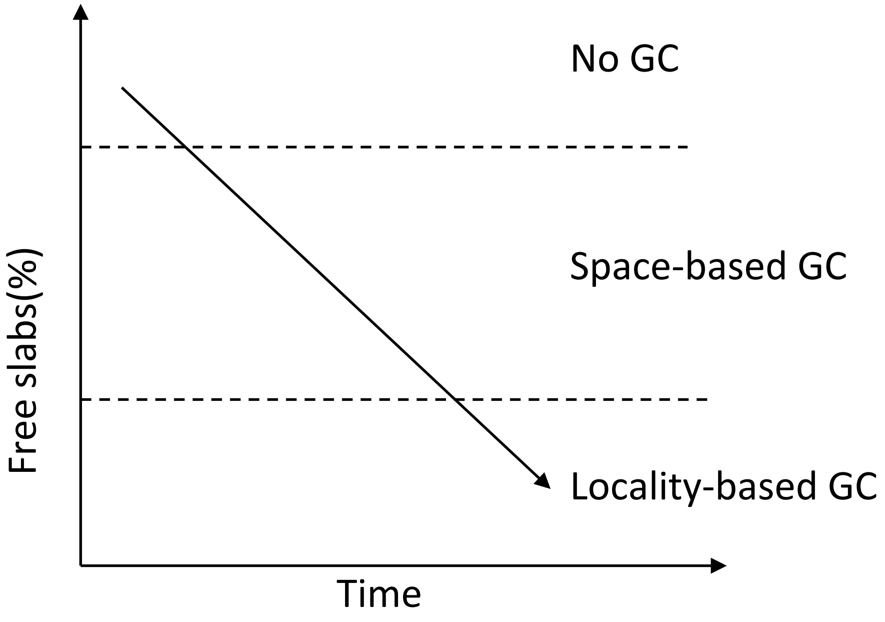
Integrated Garbage Collection



- All writes in unit of flash blocks
- Remove unnecessary device-level GC
- Application-driven fine-grained GC

Integrated Garbage Collection

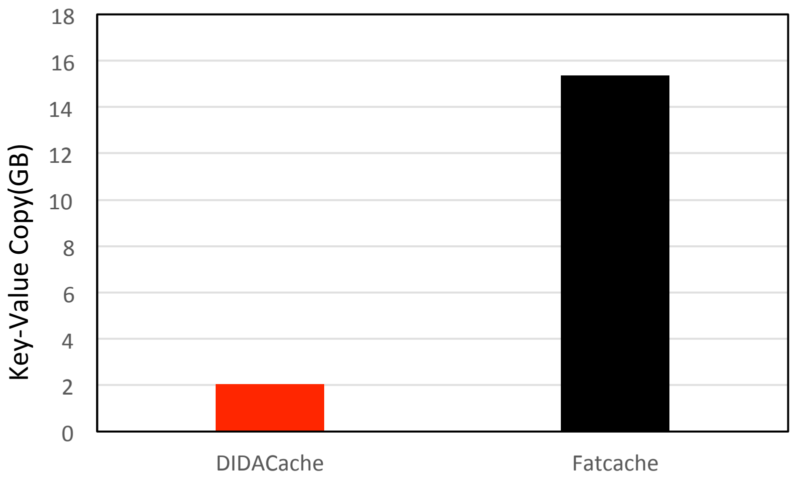
- GC is a time consuming process (key-value copy and block erase)
- Goal: retain high key-value cache hit ratio and low latency



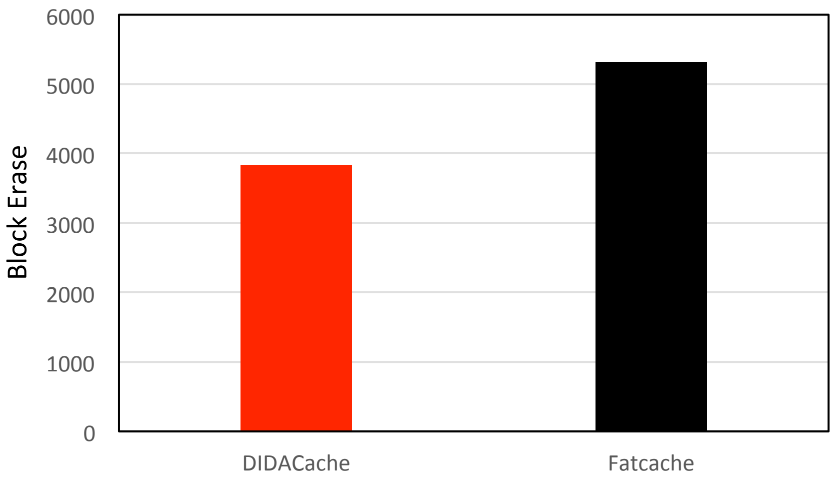
- Light traffic: Space-based GC
  - Optimize for high hit ratio
  - Select the block with the most invalid items
  - Copy valid items and erase the slab
- Heavy traffic: Locality-based GC
  - Optimize for low response time
  - Select the LRU block as the victim
  - Erase the entire slab without item copy

Integrated Garbage Collection

- Garbage collection overhead
  - DIDACache makes 86.6 % less key-value copies
  - DIDACache erases 30% less flash blocks on device



Key-Value item copy

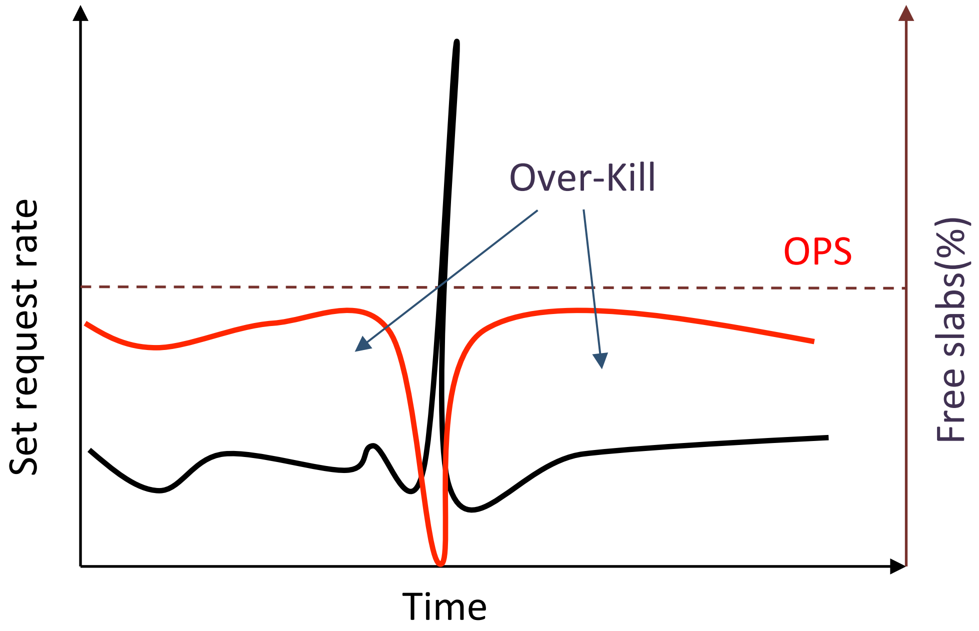


Block erase count

* DIDACache: directly collect GC overhead, Fatcache: blktrace+ssdsim

Over-Provisioning Space Management

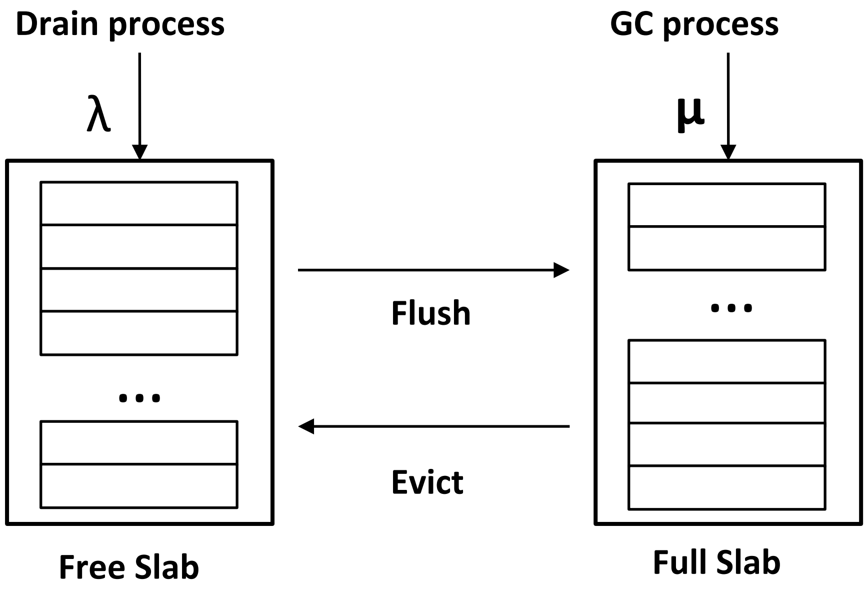
- OPS is a large (20-30%) reserved space for handling intensive writes
- Goal: maximize the usable flash space for caching and keep just enough OPS



- SSD is used as cache, not storage
  - Workload for Key-value cache is read intensive
  - 20-30% OPS is an unnecessary over-kill
- Disadvantage of static OPS
  - OPS not usable for key-value caching
  - Low hit ratio with too large OPS

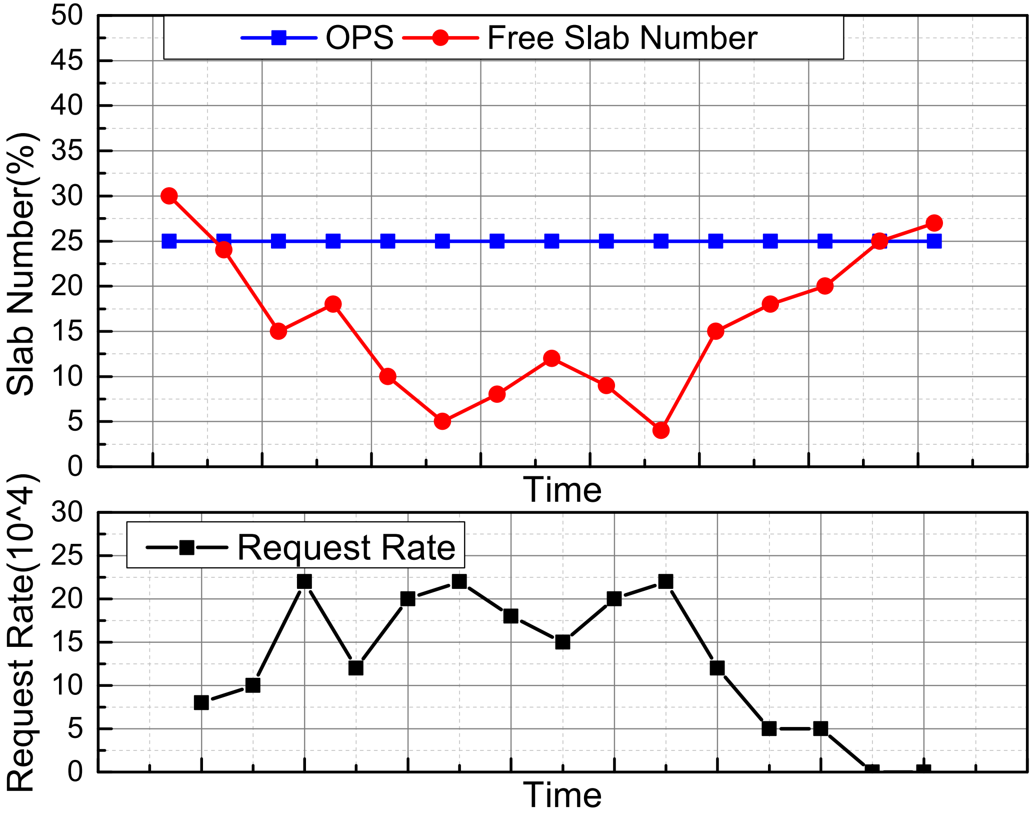
Over-Provisioning Space Management

- Queuing theory based OPS estimation
  - Drain process: a Markov process with rate λ ($\lambda = \frac{\lambda_{KV} \times S_{KV}}{S_{slab}}$)
  - GC process : a Markov process with rate μ ($\mu = \frac{1}{t_{evict} + t_{other}}$)
  - Little's law: $OPS = \lambda / (\mu - \lambda)$

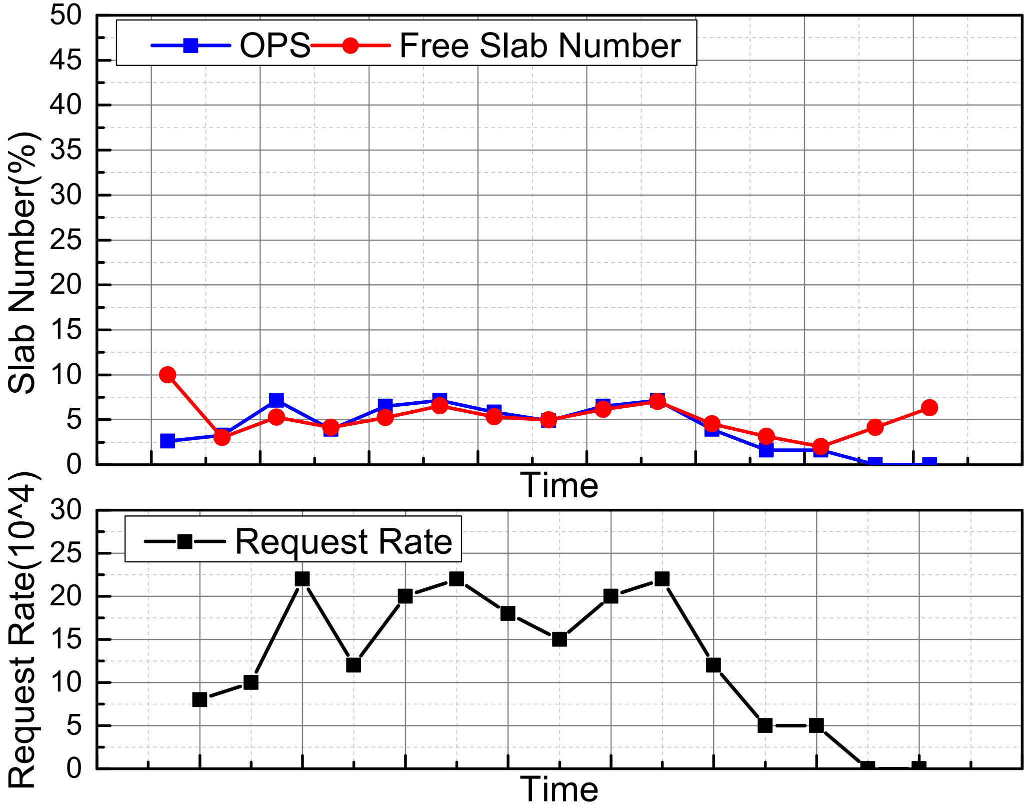


Over-Provisioning Space Management

- Over-provisioning space with different policies



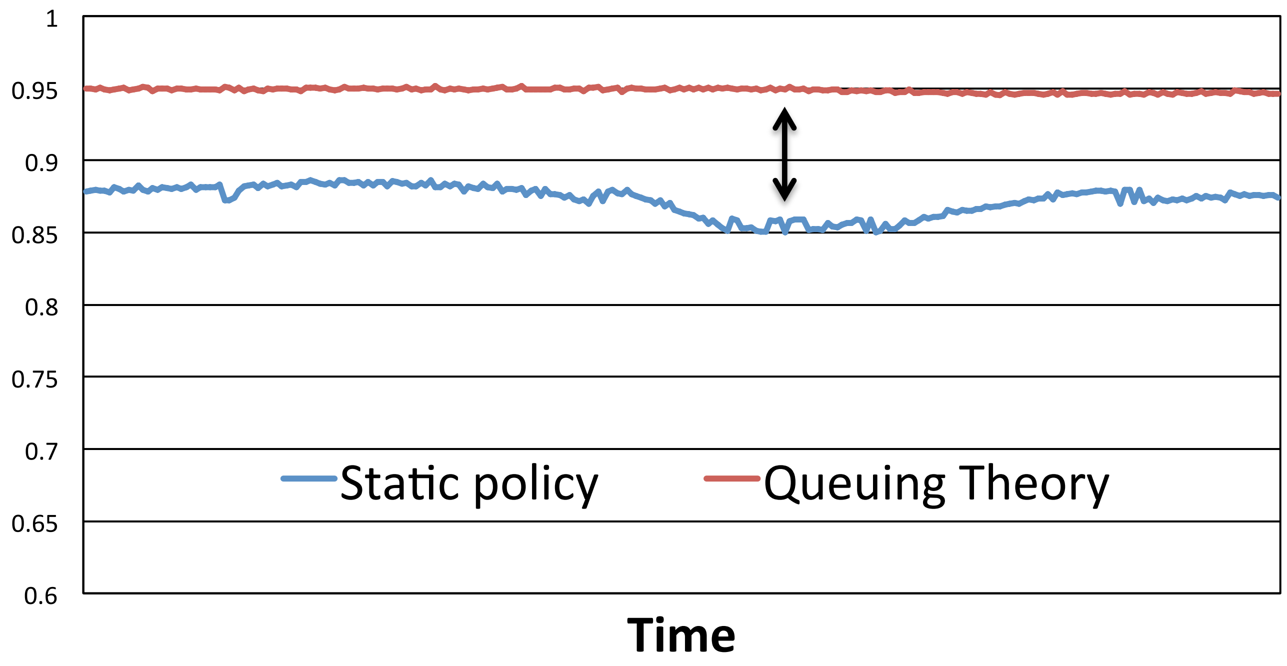
Static policy



Queueing theory policy

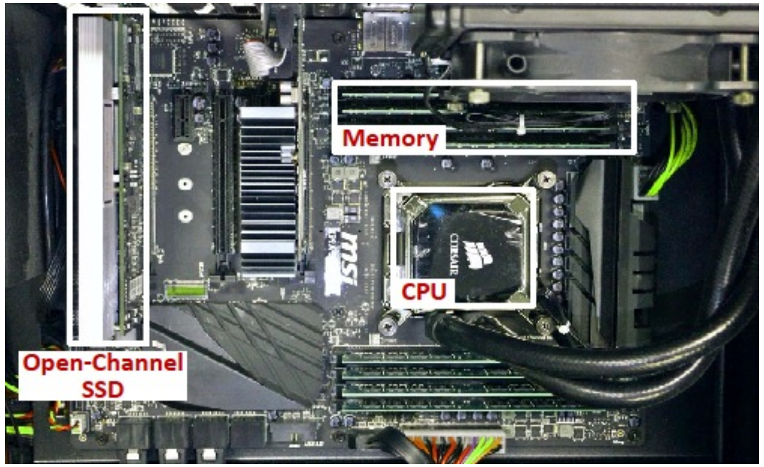
Over-Provisioning Space Management

- DIDACache improves hit ratio with dynamic OPS management



Experiments

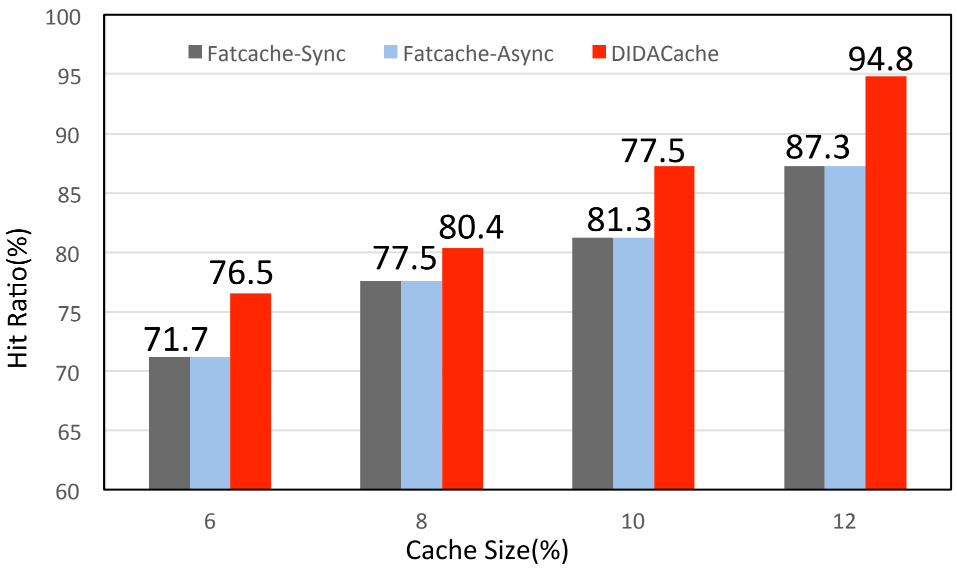
- Implementation
  - Key-value cache on Twitter's Fatcache to fit hardware
  - Schemes: Fatcache-Sync, Fatcache-Async[1], DIDACache
- Experimental Setup
  - Intel Xeon E-1225, 32GB Memory, 1TB Disk
  - Ubuntu 14.04 LTS, Linux 3.17.8, Ext4 filesystem
  - Database: MySQL 5.5
  - Workload: truncated Generalized Pareto distribution
- Storage
  - Open-channel SSD:
    - A PCI-E based with 12 channel, and 192 LUNs
    - Direct control to the device (via `ioctl` interface)
  - A conventional SSD with the same hardware configuration

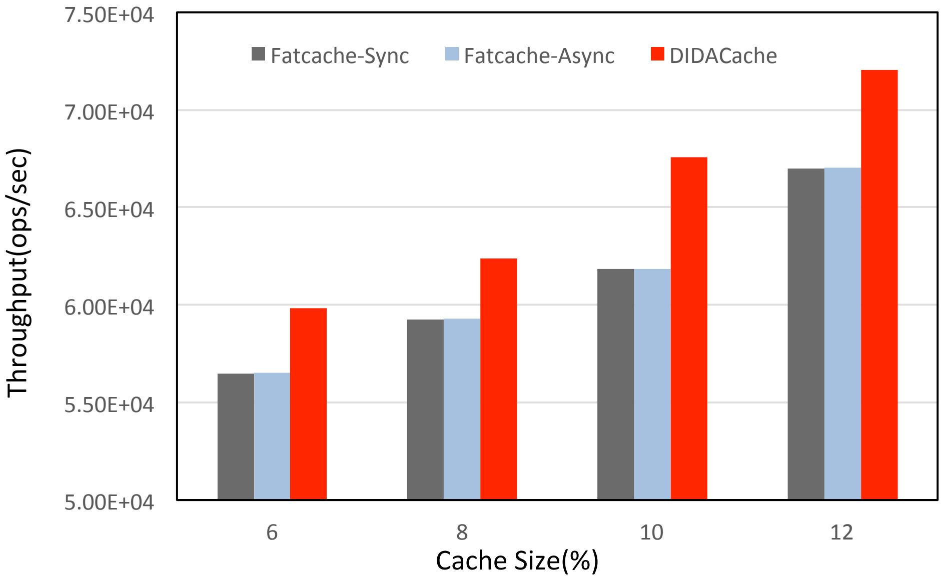


[1] Fatcache-Async. https://github.com/polyu-szy/Fatcache-Async-2017

Overall Performance in a Data-center Environment

- MySQL + Key-value Cache + Client



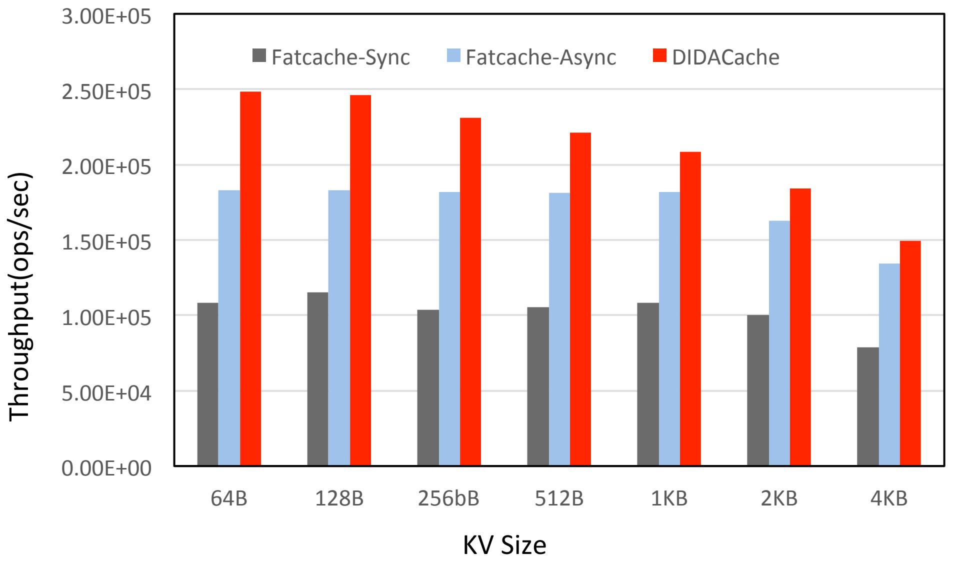


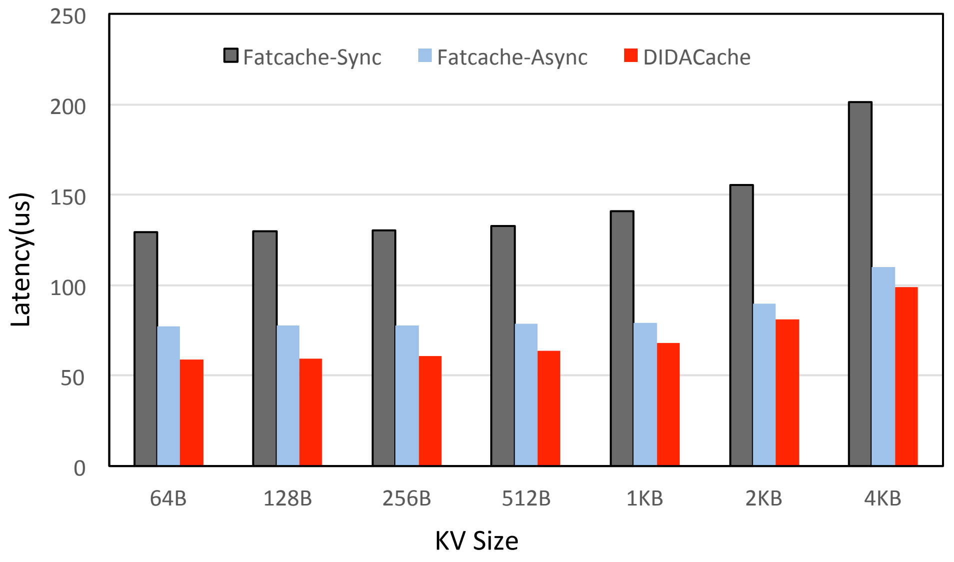
- As the cache size increases, all throughput improves substantially
- DIDACache has the highest throughput among all the three cases

*** A data-center environment running with 250GB MySQL database, 8 key-value caching servers and 32 clients**

Cache Server Performance

- Key-value Cache + Client: set operation



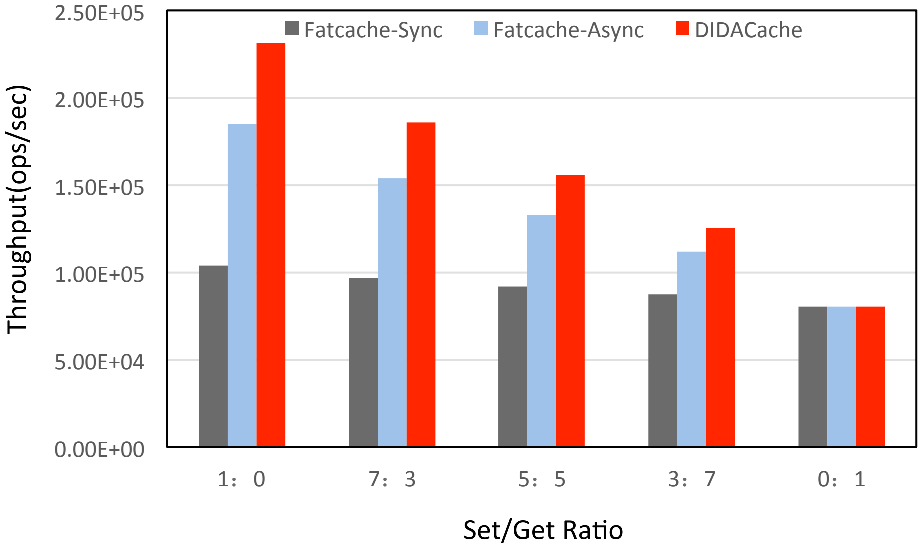


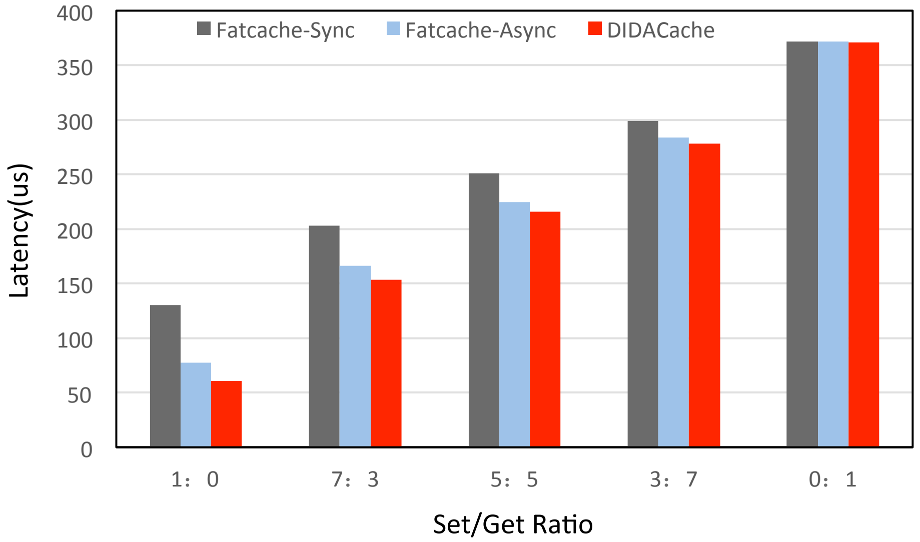
- DIDACahce achieves the highest throughput and lowest latency.
- With the item size of 64 bytes, throughput of DIDACache can be 35.5% higher than Fatcache-Async. Latency can be reduced by 23.6%.

*** Directly SET 50GB key-value items (ranges from 64Bytes to 4KB) to the cache servers**

Cache Server Performance

- Key-value Cache + Client: mixed set/get operation





- DIDACache outperforms Fatcache-Sync and Fatcache-Async across the board.
- As the ratio of GET operations increases, the related performance gain reduces.

*** Mixed set/get operations with key-value items of size 256bytes.**

Conclusions

- DIDACache deeply integrates the key-value cache system design with the Open-Channel SSD hardware.
- The prototype based on the Open-Channel SSD hardware shows that our approach can improve system performance significantly.

Thank You !

Q&A