# Flashield: a Hybrid Key-value Cache that Controls Flash Write Amplification

By: Assaf Eisenman , Asaf Cidon, Evgenya Pergament , Or Haimovich , Ryan Stutsman , Mohammad Alizadeh, and Sachin Katti

Presented by: Ali Zaidi

# Agenda

- Domain context and overview
- Problem statement
- Current State of the Art
- Proposed Solution Overview
- Analysis
- Conclusion

# Rise of SSDs in Cloud Applications

- Price per bit in SSD has steadily dropped over the last few years, more than 10x lower than that of Dynamic Random Access Memory (DRAM)
- As a result, it has become the preferred storage medium for hot data, where high throughput and low latency are desired

| | SSD+DRAM | | DRAM only | |
|---|---|---|---|---|
| | Count | Cost | Count | Cost |
| Dell 2×10 core server with 256 GB DRAM | 1 | $7,700 | 17 | $130,900 |
| Samsung 1 TB enterprise SSD | 4 | $4,800 | 0 | 0 |
| Total | | $12,500 | | $130,900 |

# At the heart of SSD: Flash

- Electricity based storage medium used in modern SSDs
- No moving part, non-volatile, and easily rewritable
- Two most common flavors: **NAND** and NOR

# Key-Value Caches

- Crucial part of modern web-scale applications, used by almost all web services (Facebook, Twitter, Airbnb)
- Very write-heavy, since small objects need to be frequently inserted, updated and evicted in the cache

# Problems with SSD for Key-Value caches (1)

- Flash storage is good for large and sequential writes, since locations in it can only be written to a few thousand times in its lifetime
- Key-Value caches, by design store small objects with (potentially) short lifetimes, which can result in frequent writes
- As a result, SSDs can wear out quickly when dealing with such workloads

# Problems with SSD for Key-Value Caches (2)

- Flash pages are grouped in large blocks, which over time can be filled with a mix of ‘valid’ and ‘invalid’ pages
- Before erasing a block, valid pages must be copied to other blocks
- This is known as Device Level Write Amplification (DWLA), which can also hinder the lifetime of the flash storage
- The small, frequent writes necessary for Key-Value caches results in increased DWLA

# Effects of Key-Value Cache on DWLA

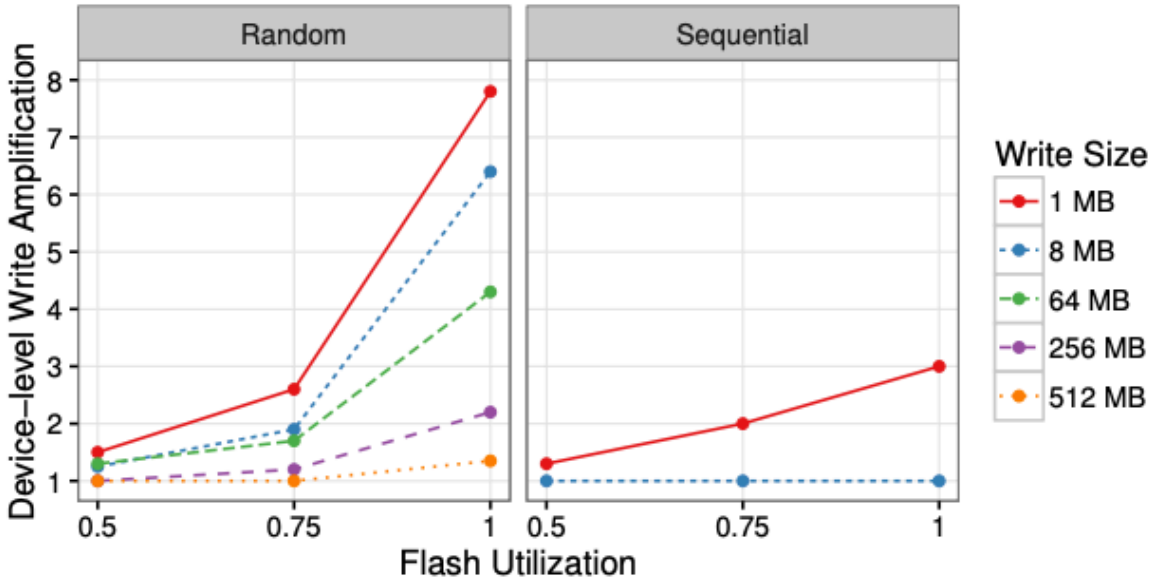



**Figure 1:** Device-level write amplification after writing 4 TB randomly and sequentially using different write sizes.

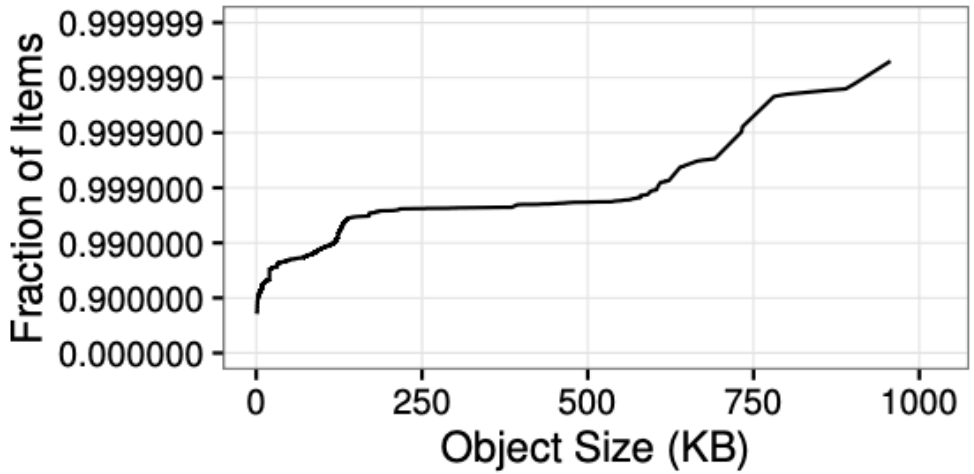
# Problems with SSD for Key-Value Caches (3)

- The secondary form of write amplification that occurs in SSDs is known as Cache Level Write Amplification (CLWA)
- This occurs as a result of trying to write to flash in large segments, in which objects removed form the SSD are re-written to it by the cache eviction policy
- The small writes of Key-Value Caches may result in both frequently and infrequently read objects to be grouped together, resulting in them both being evicted

# Effects of Key-Value Cache on CWLA

| Avg Object Size | Read / Write / Update % | Unread Writes % |
|---|---|---|
| 257 B | 90.0% / 9.5% / 0.5% | 60.6% |

**Table 2:** Statistics of the 20 applications with the most requests in the week-long Memcachier trace.



# Current State of the Art: RIPQ

- SSD-based photo cache that is currently the best at minimizing CWLA
- Deals with CWLA by storing hot and cold objects together on flash, Inserting objects that were read *k* times in the past together
- Main Idea: objects that were ready k times in the past might have a similar eviction rank

# Problems with RIPQ

- Works well for large and immutable data like photos, but in Key-Value caches, where small and frequent updates are the norm, it does not work as well
- In applications where many objects are infrequently accessed, RIPQ will desginate these objects as “cold”, most likely getting evicted before they are accessed again, increasing the CWLA
- RIPQ has no admission policy and writes all incoming objects to flash, even unread or frequently updated objects
- When the frequency of reads of an object changes, it results in addiontal writes

| | Hit Rate | CLWA |
|---|---|---|
| Victim Cache | 69.72% | 4.00 |
| RIPQ | 70.59% | 2.59 |

**Table 3:** Hit rate and cache-level write amplification of RIPQ and the victim cache policy under the entire Memcachier trace.

# RIPQ vs Victim Cache Policy: Better, but not Great

# Flashield Design

- Goal is to minimize CLWA and DLWA, while keeping a comparable hit rate
- Use DRAM as a filter before moving objects into Flash
- Use a machine learning classifier to rank objects, the highest of which are moved into flash
- Once an object is evicted, it must prove itself once again to be "worthy" of being placed into flash again

# DRAM Filter

- DRAM is the gatekeeper of which objects are written into the “precious” flash
- Ranks which objects are ”flash-worthy”, and then has the highest rated objects entered into flash
- Uses machine learning to dynamically learn for each individual applicaiton

# DRAM Filter in Action

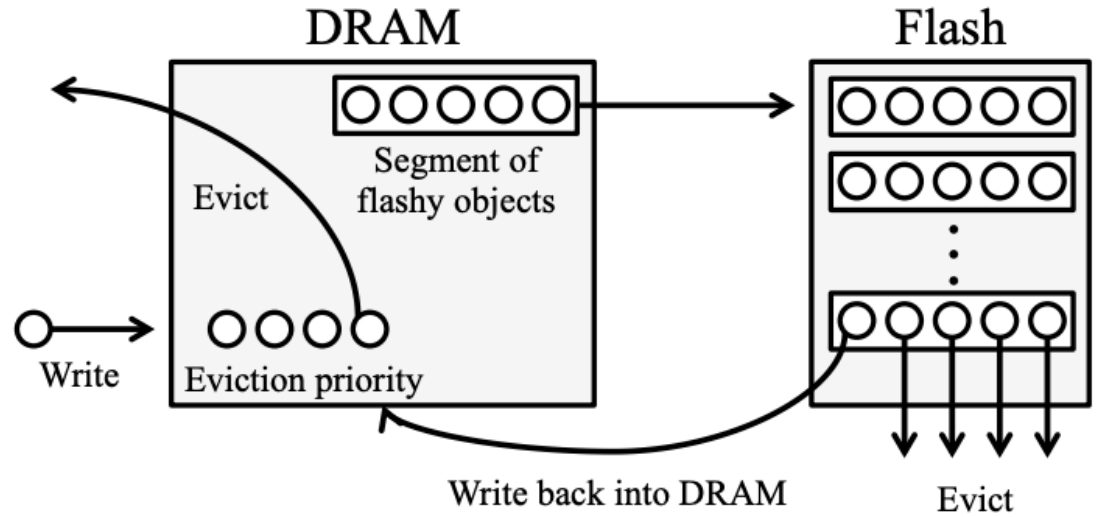



**Figure 3:** Lifetime of an object in Flashield. Objects always enter into DRAM. Objects that are a good fit for flash (*flashy* objects) are aggregated and moved into flash as a segment. The decision of whether to evict objects from DRAM or flash is based on a global eviction priority.

# Machine Learning in Flashield

- Uses a binary classifier SVM to predict “Flashiness”
- ”Flashiness” score is based on if an object will be accessed *n* times in the near future, and if this object is immutable in the near future as well
- The SVM looks at two features: number of past reads, and number of past updates

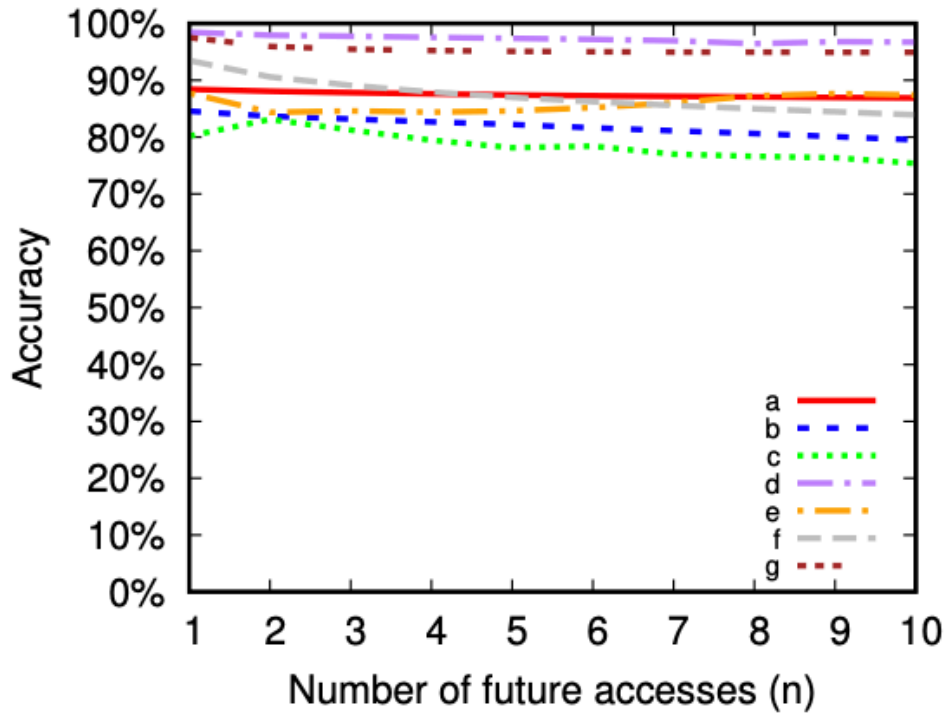



**Figure 4:** Accuracy of SVM classifier in different Memcachier applications, for predicting whether an object will be accessed at least $n$ times in the future without updates.

# Accuracy of the SVM classifier

# Further Design Considerations

- Flashield stores indexes in a hash-table in DRAM, for efficient lookup at low latency
- Key Identities are stored in Flash, as part of the object metadata
- Indexes store the segment number and a Hash Function ID (1 of 16)
- CLOCK policy used for eviction

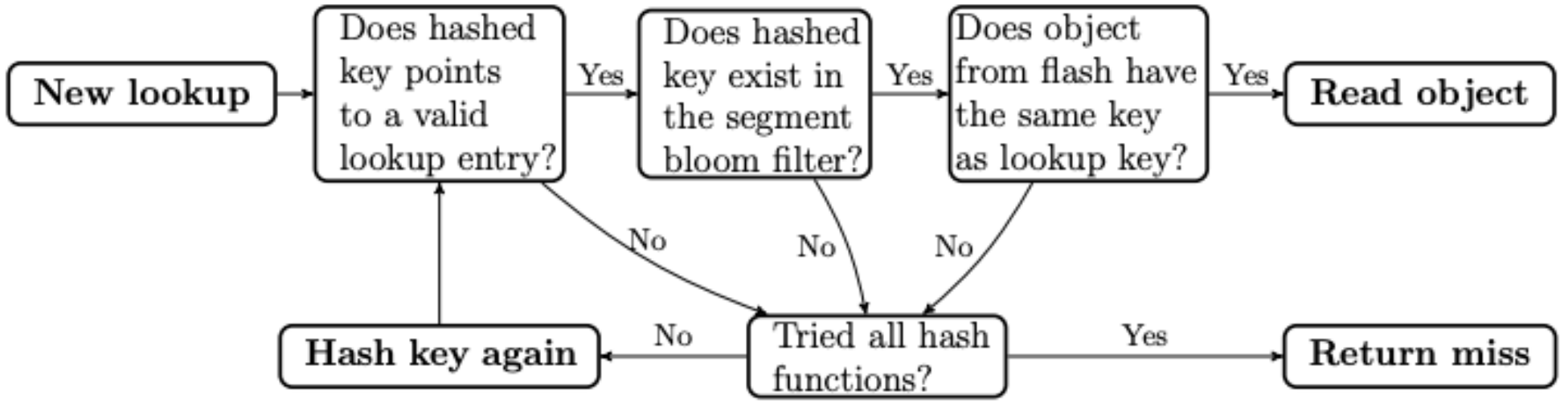



**Figure 5:** Algorithm for determining if an object exists in flash.

# Flashield's Lookup Process

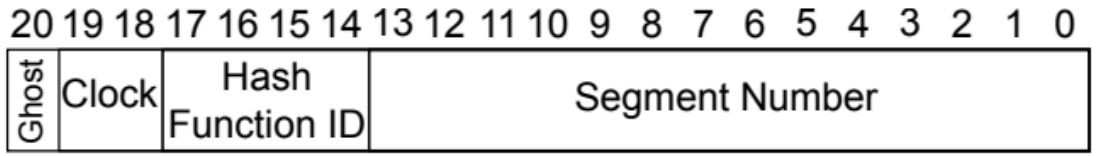



**Figure 6:** Hash-table entry format for objects stored on flash.

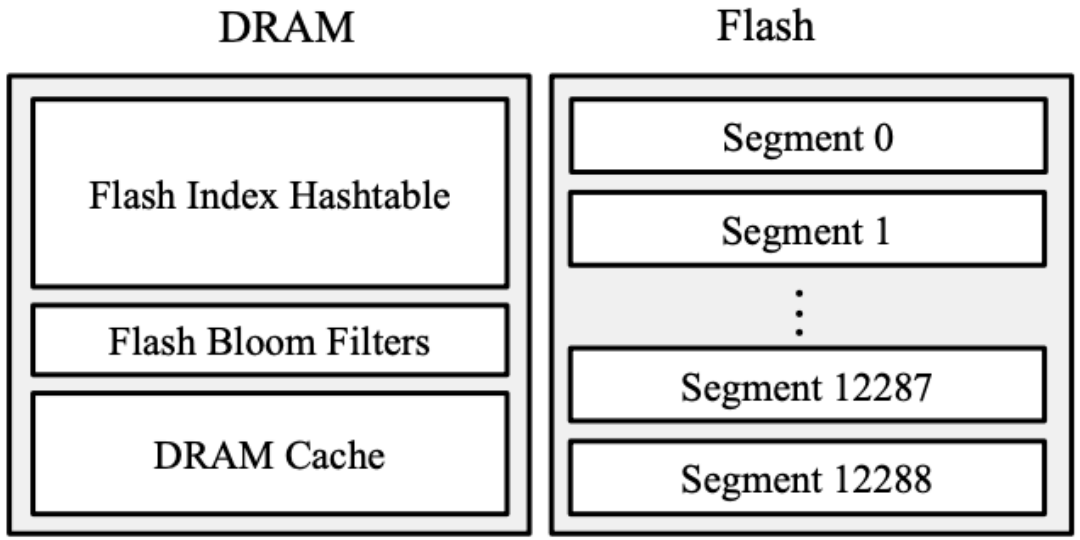



**Figure 7:** Flashield's architecture. The flash index is an in-memory hash table. The bloom filters provide fast lookups for object existence in flash, and the rest of the DRAM is a cache. Most of the cache objects are stored on flash in segments.

# Flashield Architecture

# Flashield's Allocation and Deletion Process

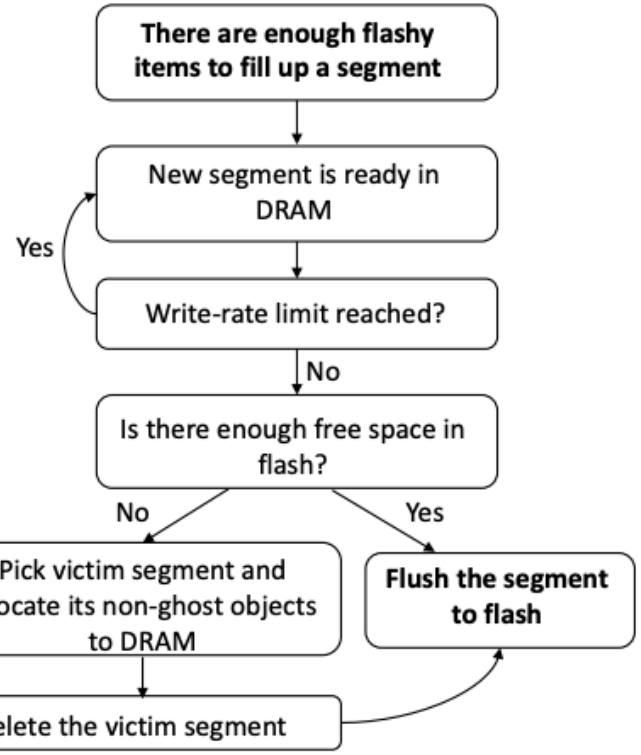



**Figure 8:** Flashield's process of allocating and deleting a segment to and from flash.

# Flashield's Evaluation

- Evaluate end to end performance compared to existing systems
- Use real-world traces of a popular Memcached service provider
- Ran a set of microbenchmarks to also stress performance to measure throughput and latency

# Evaluation Results: CWLA

- Flashield was found to have a much lower CWLA rate than RIPQ and Vicitm Cache
- Median CWLA for Victim cache: 3.67
- Median CWLA for RIPQ: 2.85
- Median CWLA for Flashield: 0.54

# Evaluation Results: Throughput and Latency

- Compared against Memcached, using 32 GB DRAM and 480 GB SSD
- Latency and throughput of DRAM hits in Flashield are very similar to that of Memcached
- Average Latency of SSD hits much higher than that of DRAM hits, but they become similar when deploying over a network
- Write throughput and latency of Flashield identical to Memcached

# Evaluation Results: DRAM:SSD Ratio

- Reduciton in DRAM may result in a drop in hit rate, since now objects don't have sufficient time to prove themselves worthy
- Significantly high hit  rates also seem to correspond to higher CLWAs as well

# Evaluation Results: Flash Utilization

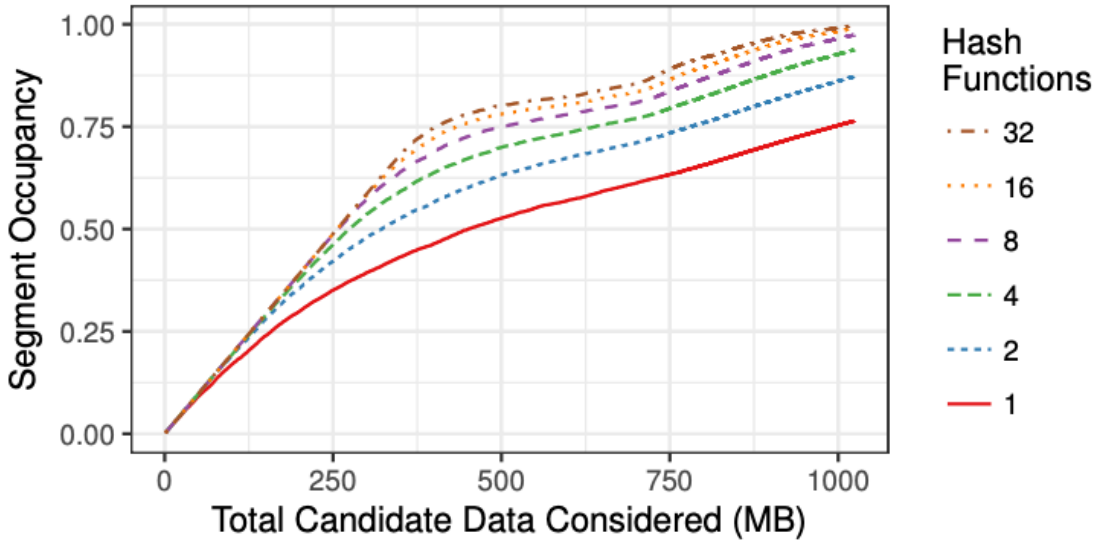



**Figure 10:** Utilization of a 512 MB segment on flash when Flashield tries to allocate space with a varying number of objects from the Memcachier trace. As Flashield tries to allocate more objects, it achieves higher utilization.

# Conclusion

- The small objects sizes and frequent eviction/update rates of key-value caches make it an interesting challenge for SSDs
- Flashield addresses this by using DRAM as a filter, coupled with a machine learning algorithm, to be more choosey about what and when to insert objects into flash
- This approach has been shown to be effective in reducing Write Amplification, the biggest problem facing other approaches to this problem

# References


- https://www.enterprisestorageforum.com/storage-hardware/flash-vs-ssd-storage-whats-the-difference.html
- https://www.usenix.org/system/files/nsdi19-eisenman.pdf