Advanced Storage Systems Project Report

Write Efficient SSD Cache Design to Improve Performance

Mohammad Javad Abdi
Abdijavad110@gmail.com
95106437

Sabihe Tajdari
Tajdari996@gmail.com
97212318

Project Summary

As SSDs become more important in storage systems, the number of diverse caching mechanisms considering different aspects of the problem will grow. One of the basic classifications mentioned in the state of the art classifies the methods in two groups named 'pull mode' and 'push mode'. The first group will consider the recent hit/miss state of data to increase the hit ratio and the second will keep hot data in SSDs long enough to reduce the number of updates. In this project we want to improve one of previous solutions to increase the hit ratio and care about the number of updates.

1. Introduction

By increasing use of data intensive applications the importance of the problem of storage systems not acceptable performance is increasing. The storage designers found that using just HDDs in the system will not solve the problem. So they tried to use other storage devices like SSDs, DRAMs and other NVMs in the system. Because of the high cost and reliability problems of the other devices like DRAM we cannot use them instead of HDDs changing the system platform completely. But there are some methods like caching and tiering which can help us to improve the performance of system using the benefit of both HDDs and other fast devices in the system. In this project we work on caching mechanisms to improve performance of storage systems. Appropriate fast storage mediums can be Volatile like DRAMs and Non-volatile like SSDs. Because of the power and cost of DRAMs we prefer to use SSDs as a cache, but they have some problems such as:

1- Limited Endurance
2- Higher cost than HDD

So we have to find an optimum request analysis and data management method to care about the first problem. The cache policy should be smart enough to increase the hit ratio in small footprints paying attention to the second problem.

The cache policy consists of two main parts:

1- Promotion
2- Eviction

The first part means the act of finding and bringing the most efficient data to the cache and the second one means finding and deleting the useless data from the cache for example when the cache is full and a new data should be promoted to cache. one of the basic ideas is using LRU queue. Because of the existing problems the correct LRU queue cannot be implemented completely. One of the methods is to bring the last recently used data at the first of queue and evict the last data of queue from the cache. this method is not good for SSD caches in storage systems because of the limited endurance of SSDs and the data access patterns in storage systems. For example, pay attention to this data flow:

43,40,39,38,37,28,7,6,5,2,28,4,3,40,28, …

If the cache size and so the LRU size equals 4 the most frequent accessed data 28 hits just one time and the number of writes to SSD equals 17 so it won't cause any performance or endurance improvement. By using a simple access count queue instead of simple LRU the number of hit ratio, performance and write efficiency can be improved.

In this project we want to propose an adaptive method to improve the performance of system caring about the endurance of SSD cache. to do this we first analyze some basic ideas using LRU queue and then want to improve them. In the next part the most important basic idea is mentioned and after that we want to speak about our idea and improvements.

2. Previous Works

Some of the most related articles are:

1) WEC [2]
2) BRP [1]
3) LARC [3]
4) Elastic queue[4]
5) RECA [5]

Some of this works just care about performance and some other try to pay attention to the endurance of the SSD cache. the most important basic idea divides the methods into two groups:

1- Pull mode
2- Push mode

The first group are the articles which try to store and monitor any data in the cache and bringing a new data to the cache will evict some data from the cache.

The second group will just bring efficient data to the cache and promote a new packet to the cache after eviction phase, not vice versa.

In this project we concentrate on the second method because it's better for endurance improvement. WEC is one of the articles belonging to this group. It uses LRU

like queue and two threads to bring and save useful data in cache. when a new read request is issued, the system will store it to the DRAM. If the cache resident data is not accessed for an optimized number of total issued requests the eviction thread will start to work. After that the promotion will be done and some data from the LRU queue will be evicted form DRAM or promoted from HDD to the SSD. The SSD cache is just read cache and any write request will evict the data from cache and LRU queue.
In this project we want to improve it, at the end the latency of system will be optimized, but right now the hit ratio and number of efficient writes is important for improvement measurement.

Another state of the art is BRP [1]. Its contribution related to us is using popularity instead of access count for data measurement. It uses two queues. One of them is basic LRU and the other is popularity based which is updated in time intervals. In the promotion phase a new data will be promoted to the cache from this queue. By setting optimum intervals and mixing this idea with WEC the performance and endurance of system will be optimized.

The state of the art RECA uses machine learning to improve performance of system by bringing useful data to cache, also it can improve the gain of endurance by increasing write efficiency but it does not count number of writes. We will use it in on future works.
This is the table of previous works rating:

Table 1: comparison of some previous methods

State or the art	Performance	Endurance	Endurance gain
WEC	3	1	1
BRP	2	3	2
LARC	3	2	3
LRU	5	5	5
RECA	1	3	1

Paying attention to this chart we can understand that each work needs to be improved in at least one part performance or endurance. Another achievement is that at least one optimum point can be found during performance and endurance improvement.

In next parts we will understand that both ideas can be improved to find an optimum point with efficient number of writes to SSD and performance improvement. Because in some situations they are in contrast.

3. Codes and Implementations

Our implemented codes divide into three categories: simulators, implementations, and trace-related codes. We used simulators to obtain our motivational results, and implementations were used to compare proposed method with previous ones. Trace related codes were formed to ease the process of simulation and calculating some basic statistical features of each trace.

3.1. Simulators

To test our methods more efficient, we first simulate the results and then go through real implementation. Our simulator is able to run uniformed trace and get simulated results. These results consist of hit ratio, write efficiency, writes on each device, RAM hits, SSD hits, simulated latency, and so on. This simulator now is able to simulate mentioned methods in WEC, our promotion threshold idea, WEC with block popularity (mentioned in BRP) idea for promotion, and frequency-based promotion. It is also able to simulate another state-of-the-art proposed method called Clock-DNV.

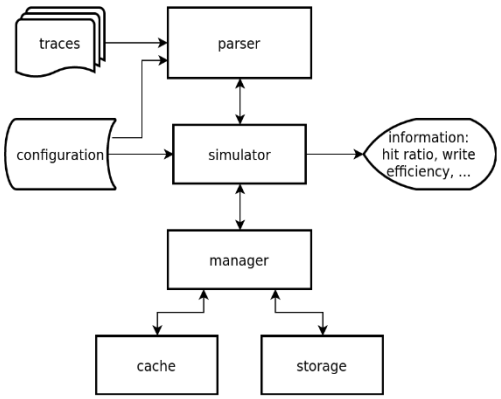


Figure 1: simulator architecture

3.2. Implementations

To test our simulations and be sure of the correct functionality of those, we had to test our proposed method on a real system. Therefore, we wrote a forwarder module that make an abstraction of physical layer and make us enable to read and write directly on raw devices. It also handles cache mapping tables, and does the translation automatically; as a result, above layer see read, write, move x from device a to device b. then, we integrate these functions inside our simulator. This way we overcame the overhead of re-implementing our method and get results on real systems.

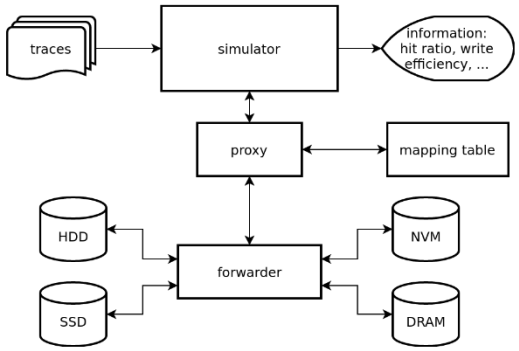


Figure 2: our implementation modules architecture

3.3. Trace-related codes

We examined various kinds of traces: snia, filebench, systor, postmark, and tpcc; as a result, it was necessary to have some codes to handle the difficulty of different arrangement in trace.

a. trace parser

Each of mentioned traces had their own arrangement for elements of requests. This diversity leads us to implementing a parser which can get any kind of trace and convert its arrangement to a user-defined uniform one. Other useful optional feature of this code is scaling time and address (multiplying to a user-specified scale-factor). It is also able to break each big request to smaller ones so that total size of smaller ones be same as original big request–this will help caching mechanism to handle requests more conveniently.

b. trace statistics

After having uniform traces, we had to find our candidates. This means we had to have an overview of its statistical features. To solve this issue, we wrote trace statistic module. For each trace, this module finds its total requests count, biggest address, latest request, read percentage, mean IOPS, and burst times. Due to these features we had found our candidate traces and obtain our results.

c. popular finder

After getting our results, we saw contradictory results on traces that seems to have a similar statistical feature. To understand why this happens, we wrote a code to find blocks with requests more than a given threshold, count their read and write requests, and draw some plots to understand this information.

4. Proposed Method

Previous work has 4 main shortcomings:

1- Just read cache
2- Simple cache update policy
3- Insufficient number and variety of workloads
4- Just simulation and no implementation

The main goal of this project is to improve the basic solution by concentrating on these 4 main parts. For the first one we first measure read to write ratio for in queue requests and just evict not appropriate data from cache when the request type is write for in queue or in cache data.

For the second one we set an optimum threshold for promoting the requests to the cache and we try to mix the existing ideas to find a better parameter to improve hit ratio and write efficiency at the beginning and the latency at the end.

For the third one we have tested the basic solution and new idea on different workloads with different parameters like read to write requests ratio. They are listed below:

Table 2: traces read to write ratio and total requests count

workload	RWR	count
rsrch0	11%	3m
mds1	98%	23m
mds0	29%	3m
hm1	93%	2m
hm0	32%	8m
src1_2	19%	4m
src2-0	12%	1.7m
src2-1	99%	2m
src2-2	37%	4m
stg0	31%	6m
stg1	93%	22m
ts0	26%	4m
wdev0	27%	2.6m
wdev1	0%	1k
wdev2	0/09%	300k
wdev3	19%	900
web0	59%	7m
web1	84%	1m
web3	62%	3k
tpcc	52%	3m
filebench	75%	1.8m
SYSTOR/2016031907-LUN0	96/96	95k
SYSTOR/2016031907-LUN1	67/47	27k
SYSTOR/2016031907-LUN2	96/1	254k
SYSTOR/2016031907-LUN3	96/73	91k
SYSTOR/2016031907-LUN4	94/41	31k
SYSTOR/2016031907-LUN6	92/85	184k

SYSTOR/2016031908-LUN1	86/49	5.3m
SYSTOR/2016031908-LUN2	90/44	9.7m
SYSTOR/2016031908-LUN3	87/6	8.6m
SYSTOR/2016031909-LUN0	39/88	5m
SYSTOR/2016031909-LUN2	41/48	5.9m
SYSTOR/2016031909-LUN3	38/44	6m
SYSTOR/2016031909-LUN4	37/54	5.5m

For the last improvement point we used a system with this architecture to test and measure the results:

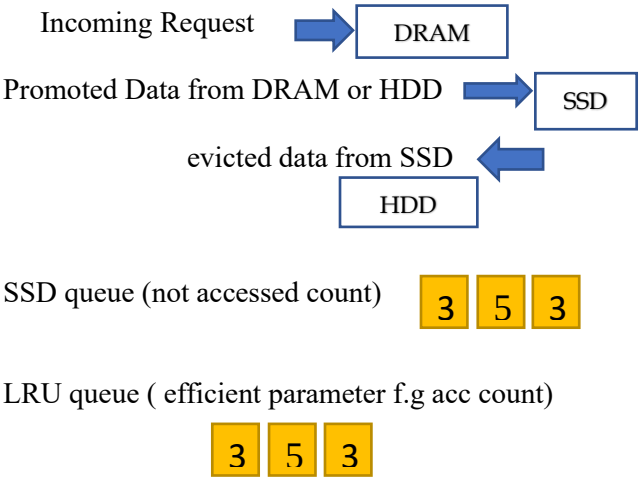


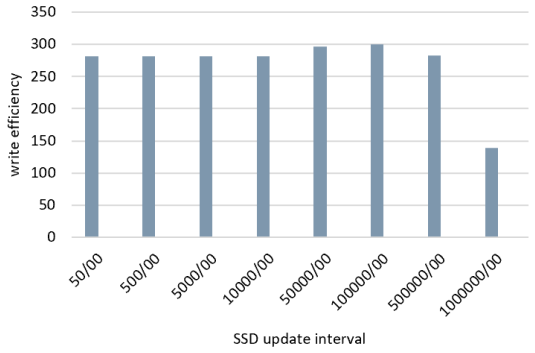
Figure 3: test system architecture

5. Results and different new ideas testing

The proposed architecture has two independent threads for incoming requests and cache updating and it has different intervals and thresholds which can be optimized. In this section we will see the testing results of this ideas one by one.

First of all, we wanted to find the optimum point of SSD update interval for the tested workloads, the results are in Figure 4.

As we see 100000 can be set as an optimum point for the tested workload, considering the hit ratio.



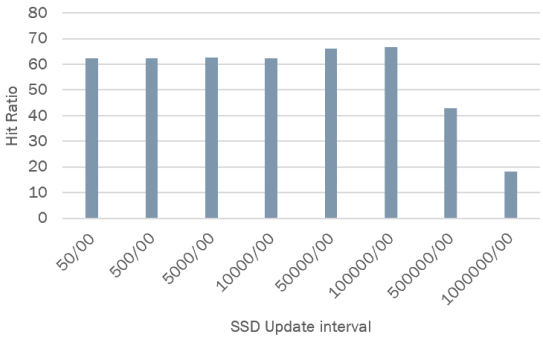


Figure 4: SSD update interval effect on hit ratio and write efficiency

After that we wanted to find the optimum promotion threshold for each workload, the results are sketched below:

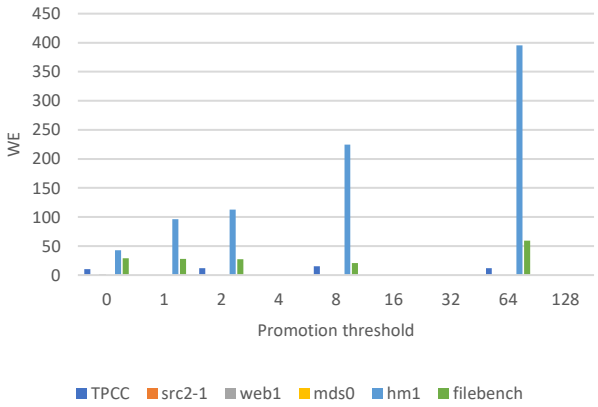


Figure 5: effect of promotion threshold on write efficiency

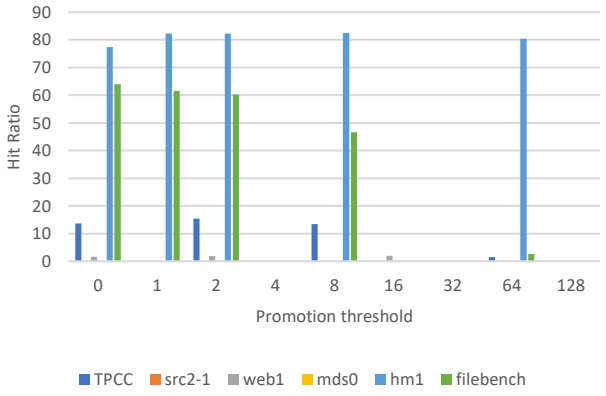


Figure 6: effect of promotion threshold on hit ratio

These results are for running workloads one by one, because the cache is just write cache the benefit of each workload load is scaled by its R/W ratio but the universal optimum point can be set between 2 and 16. The tests should be completed. After that we tested different workloads with different promotion policies. In this tests

we tried to find the optimum cache to storage size ratio. The results are mentioned below:

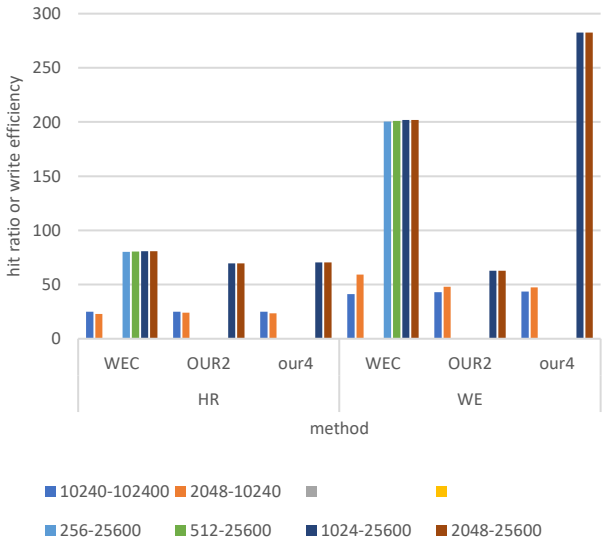


Figure 7: cache size effect on write efficiency and hit ratio

As we see the tow last tests have better results than others. After that we tried to test better parameters for data promotion.
Some results are mentioned in the attached chart.

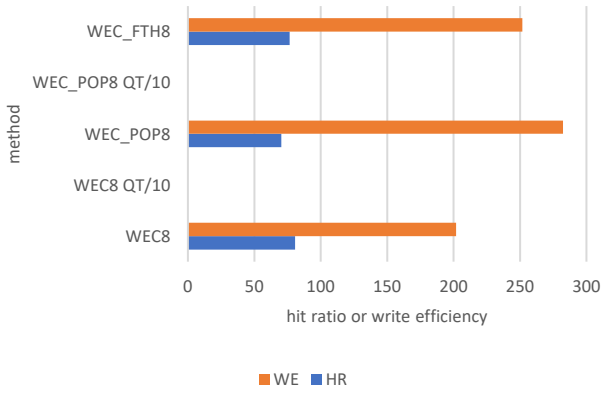


Figure 8: promotion alternatives. FTH stands for frequency threshold (we used blocks access frequency). POP stands for popularity (we used block popularity to find candidates which we want promote to cache).

As we see in the last part considering read to write ratio for eviction can improve the write efficiency and hit ratio, by finding an optimum point.

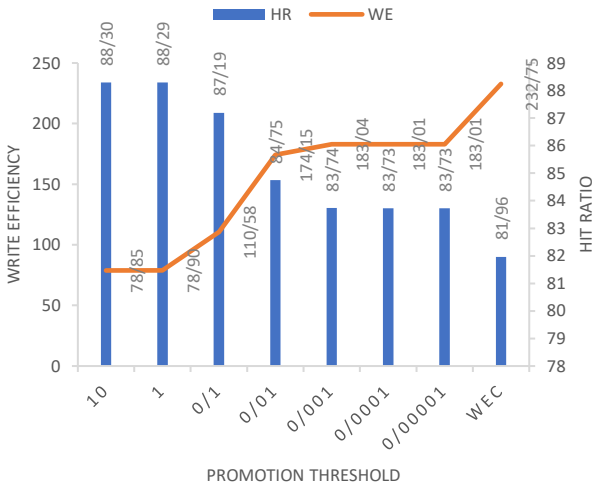


Figure 9: different promotion threshold effect on write efficiency and hit ratio

6. Future Works

This part is under revision. But at the end we want to find the effect of each part on latency, first using a formula for prediction and then real implementation.

$$latency \cong miss \times HDD_{latency} + hit \times SSD_{latency}$$

Equation 1: latency estimation formula

7. Project Overview

Table 3: project progress overview

task	% Done
state of the art analysis	100%
State of the art replication	100%
Idea improvement	100%
Idea simulation	99%
Testing and comparison	99%
Idea implementation	80%
Other references simulation	30%

8. References

[1] Ryou, Y., Lee, B., Yoo, S., & Youn, H. Y. (2015). Considering block popularity in disk cache replacement for enhancing hit ratio with solid state drive. In 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD) pp. 1-6.

[2] Chai, Y., Du, Z., Qin, X., & Bader, D. A. (2015). WEC: Improving durability of SSD cache drives by caching write-efficient data. IEEE Transactions on computers, 64(11), 3304-3316.

[3] Huang, S., Wei, Q., Feng, D., Chen, J., & Chen, C. (2016). Improving flash-based disk cache with lazy adaptive replacement. ACM Transactions on Storage (TOS), 12(2), 8.

[4] Ye, F., Chen, J., Fang, X., Li, J., & Feng, D. (2015, August). A regional popularity-aware cache replacement algorithm to improve the performance and lifetime of SSD-based disk cache. In 2015 IEEE international conference on networking, architecture and storage (NAS)pp. 45-53.