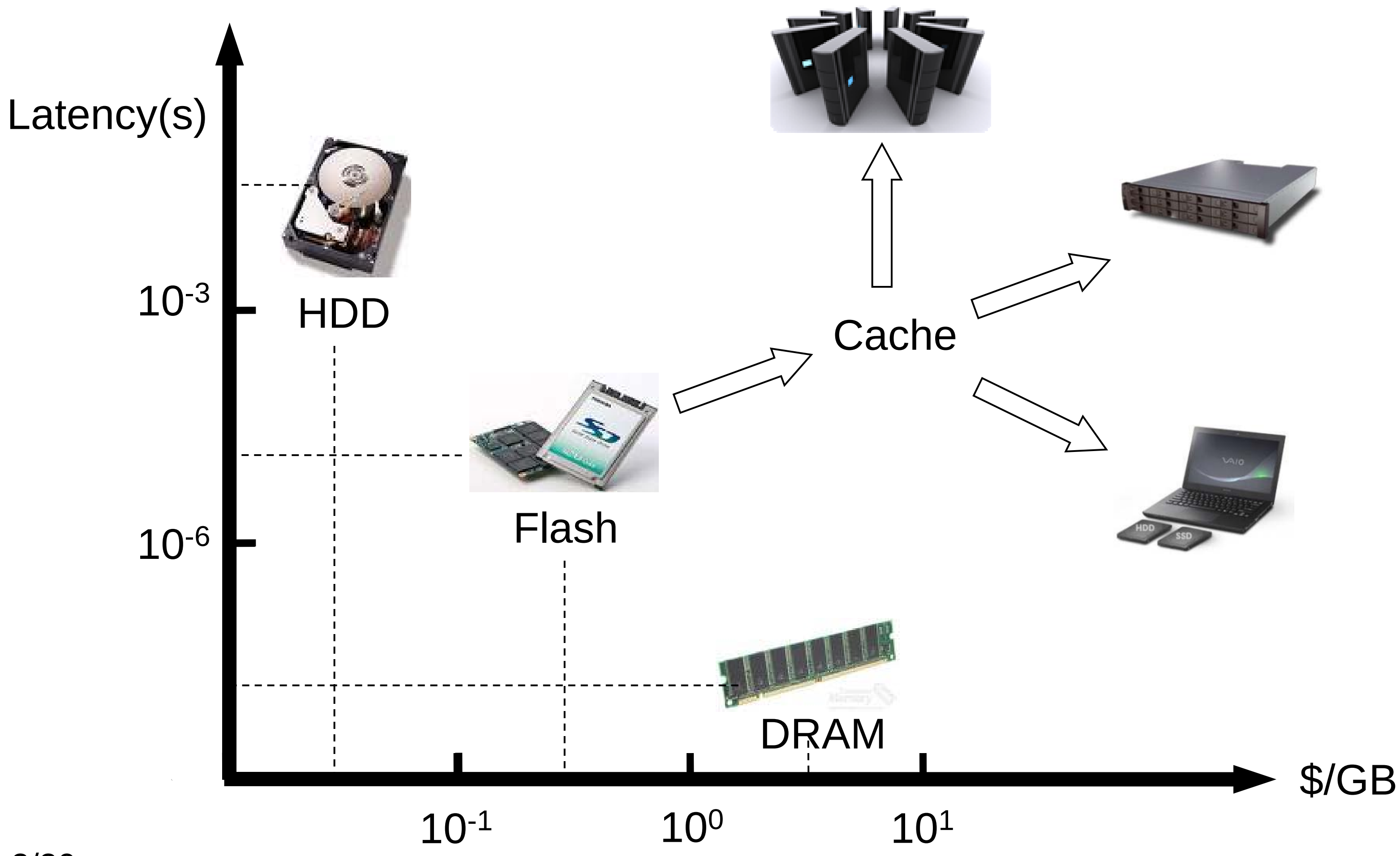
Improving Flash-based Disk Cache with Lazy Adaptive Replacement

Sai Huang, Dan Feng
Wuhan National Lab for Optoelectronics &
School of Computer, Huazhong University of Science and Technology
Wuhan, China

Qingsong Wei, Jianxi Chen, Cheng Chen
Data Storage Institute, A*STAR, Singapore

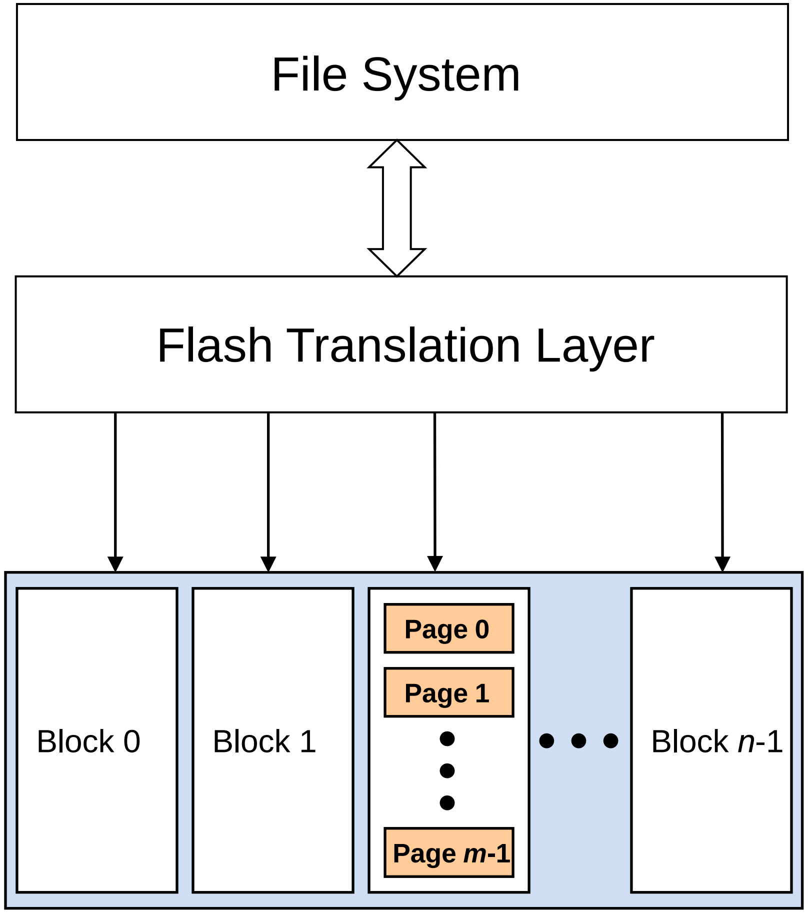
MSST, May 10, 2013

Flash Memory



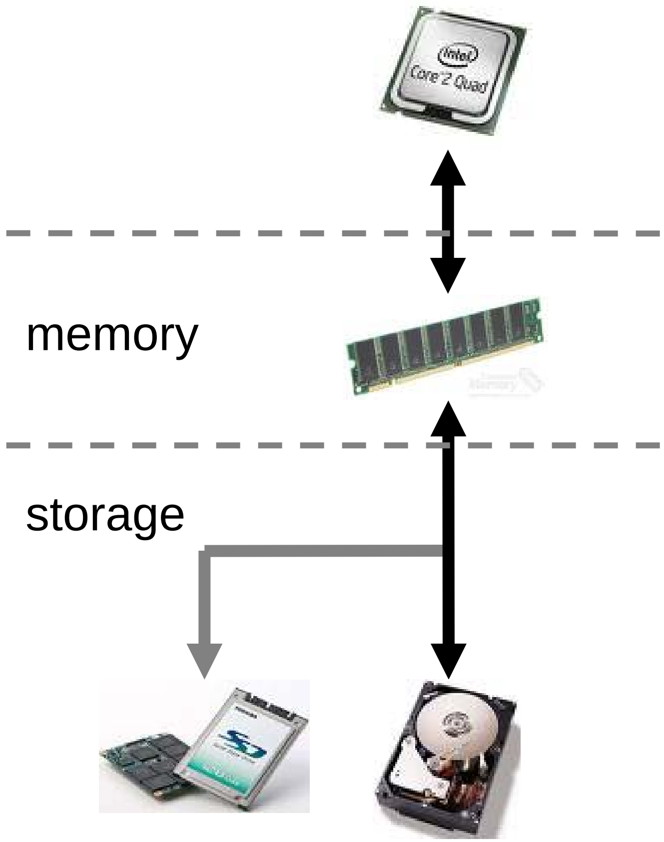
Solid-State Drive

- NAND Flash
  - read/write in pages
  - erase in blocks
  - erase before write
  - 1K ~ 100K P/E cycles
- Flash Translation Layer
  - hide the out-of-place update behaviour
  - incurs extra writes



SSD-based Disk Cache

- Performance Challege
  - second level cache
  - LRU is insufficient
- Endurance Challenge
  - write amplified by cache replacements
- Example: scan
  - flush out popular blocks
  - unneccessary writes

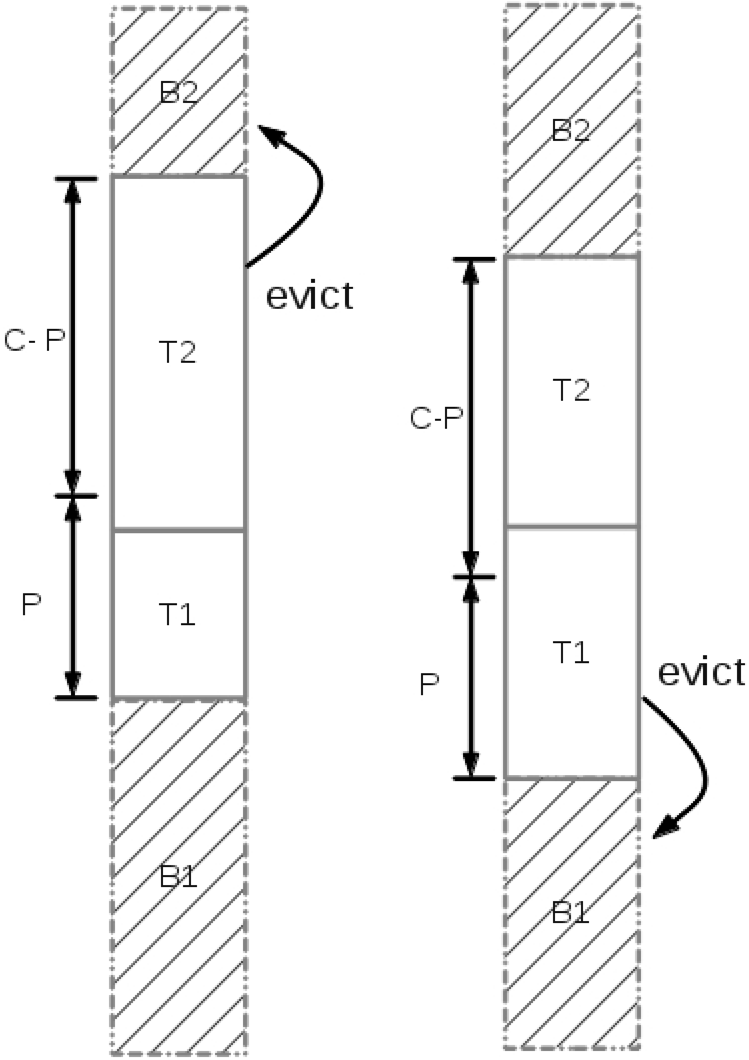


Cache Algorithms

- Objective: improve hit rate
- Temporal locality → LRU
  - simple, low-overhead
  - vulnerable to scans, loops
- Skewed popularity → LFU
  - scan resistant
  - need to deal with stale blocks
- Recency + Frequency
  - EELRU, FBR, 2Q, LRFU, LIRS, MQ, ARC, ...
  - evict seldom accessed blocks earlier

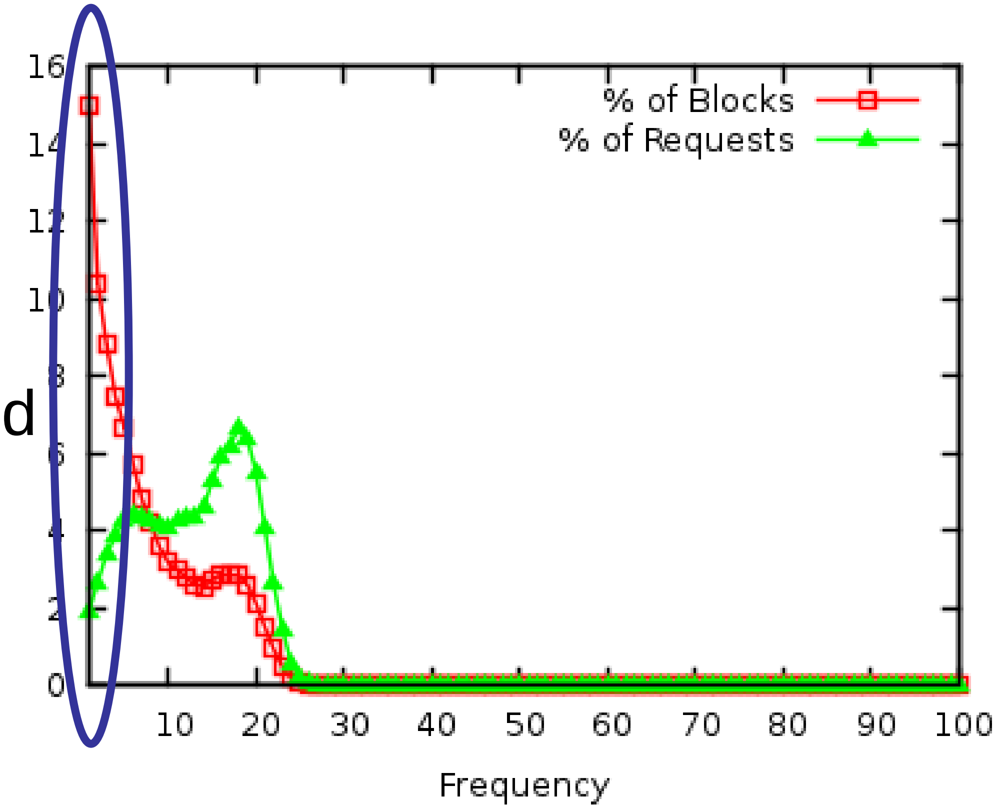
Cache Algorithms

- ARC
  - T1: one-time accessed blocks
  - P : maximal length of T1
  - blocks are evicted from T1 when |T1| > P
  - B1, B2 as feedback for tunning P



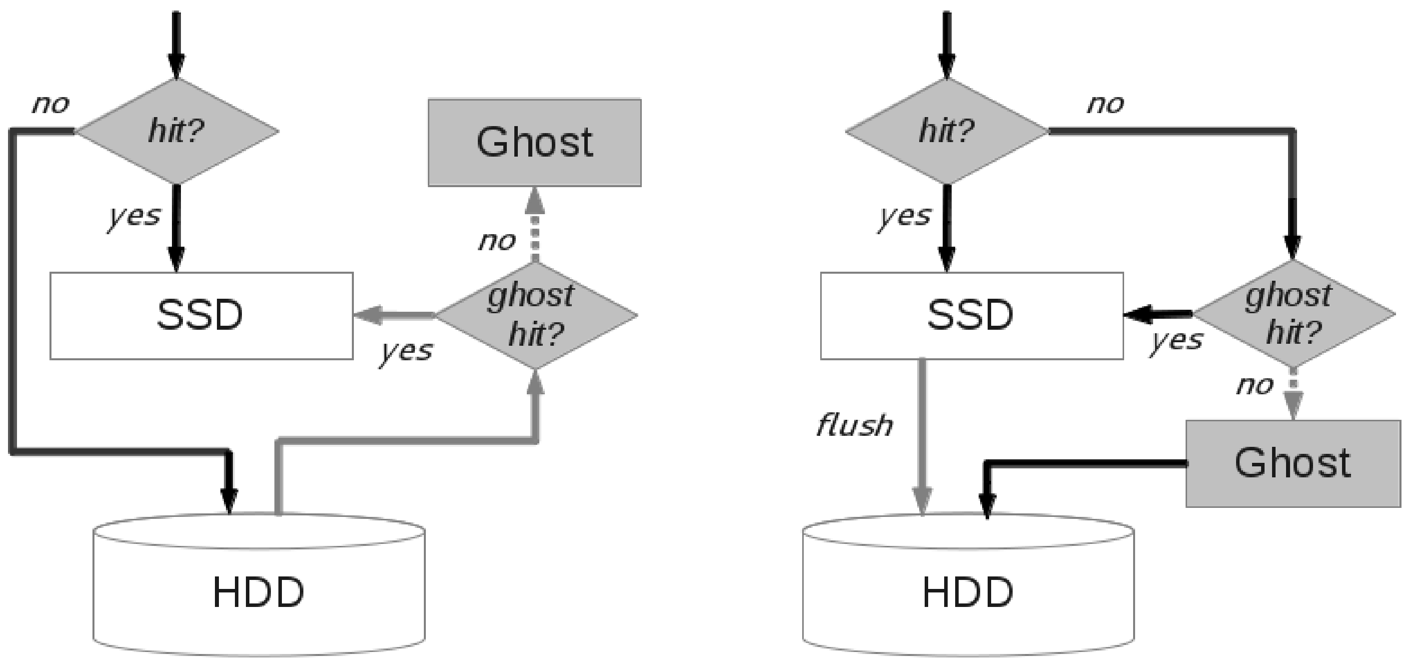
Cache Algorithms

- LRU
  - low hit rate
  - unnecessary writes
- ARC
  - evict them from T1
  - performance improved
  - endurance ignored
- LARC
  - keep them out!



LARC

- Lazy Adaptive Replacement Cache
  - keep seldom accessed blocks out
  - use a ghost cache as filter



LARC

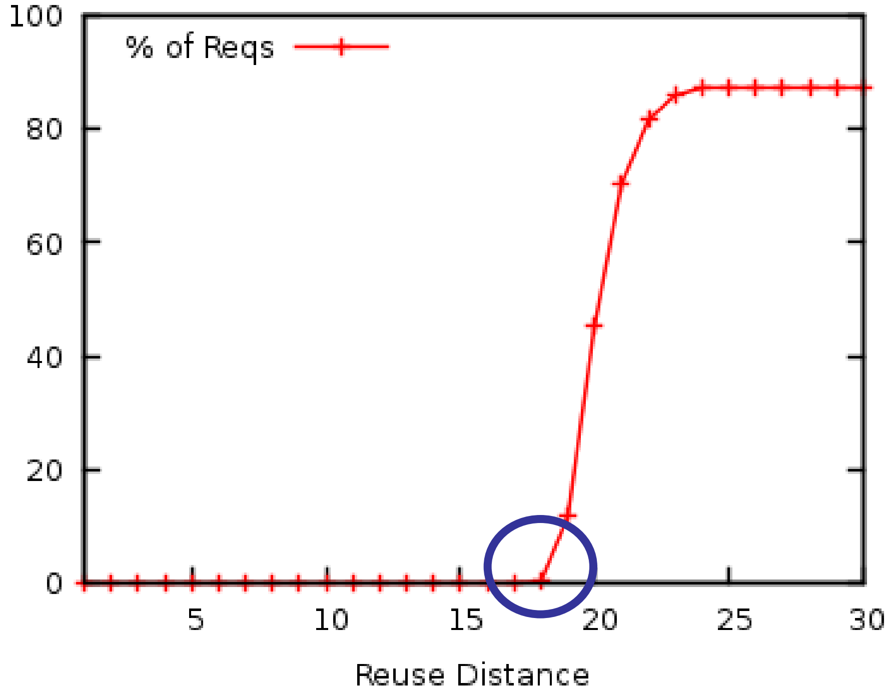
	LARC		LRU
1	0, 1, 2, 3		0, 1, 2, 3
5	1, 0, 2, 3		1, 0, 2, 3
3	1, 0, 2, 3	5	5, 1, 0, 2
4	3, 1, 0, 2	5	3, 5, 1, 0
3	3, 1, 0, 2	4, 5	4, 3, 5, 1
1	3, 1, 0, 2	4, 5	3, 4, 5, 1
0	1, 3, 0, 2	4, 5	1, 3, 4, 5
4	0, 1, 3, 2	4, 5	0, 1, 3, 4
0	4, 0, 1, 3	5	4, 0, 1, 3
3	0, 4, 1, 3	5	0, 4, 1, 3
6	3, 0, 4, 1	5	3, 0, 4, 1
1	3, 0, 4, 1	6, 5	6, 3, 0, 4

Block 3 is mistakenly replaced by LRU!

	hits	replacements
LARC	8	1
LRU	6	6

LARC

- Self Tunning
  - ghost cache size(Cr) is the "throttle"
  - minimal reuse distance
- Adjust C_r
  - hit: Cr -= C / (C - Cr)
  - miss: Cr += C / Cr
  - $C_r \in [0.1*C, 0.9*C]$



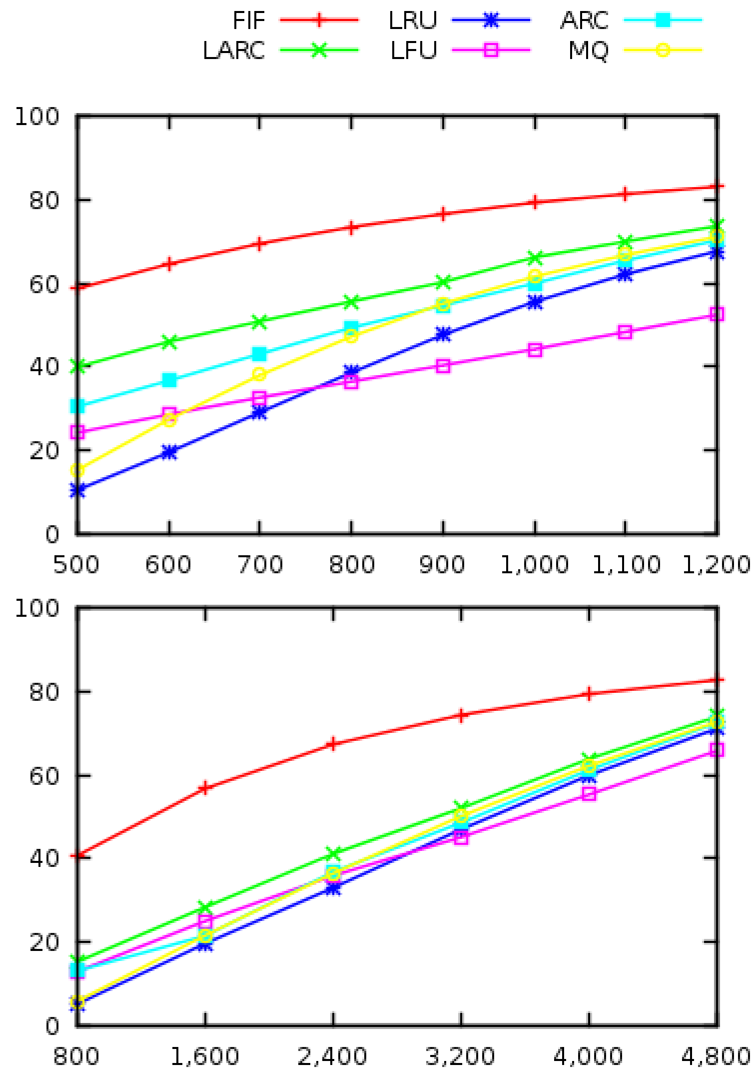
Simulation

- 4 real-life traces + 1 synthetic trace
- Compared with LRU, LFU, MQ, ARC, and FIF
- measure both hit rate and cache write traffic

	Unique Blocks Accessed (×1, 000)			Requests (×1, 000)		
	Read	Write	Total	Read	Write	% of Read
websearch	2,223	0.034	2,223	17,253	2	99.99
ads	5,408	129	5,535	14,089	348	97.59
webvm	353	248	549	3,116	11,177	21.80
homes	3,490	1,299	4,569	4,053	17,110	19.15
ws_con	3,622	0.082	3,622	33,660	4	99.99

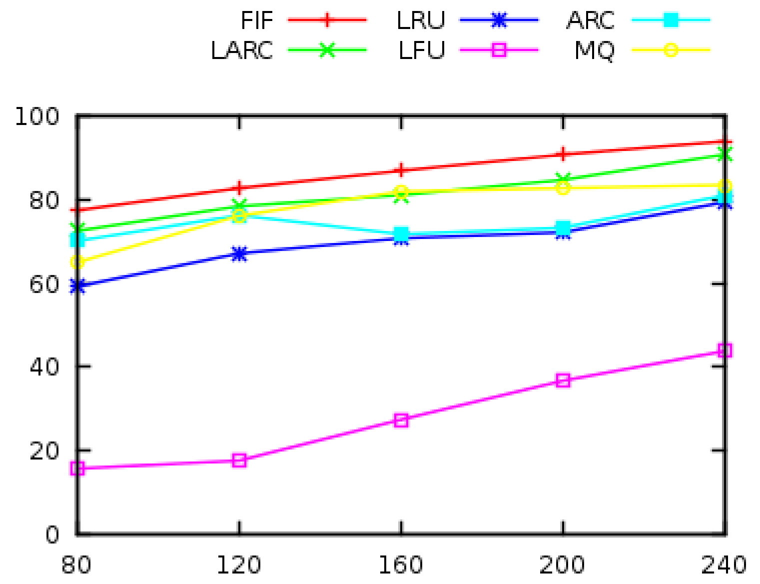
Simulation

- Hit Rate(*websearch*)
  - ↑9 ~ 277 % (v.s. LRU)
  - ↑5 ~ 31 % (v.s. ARC)
- Hit Rate(*ads*)
  - ↑4 ~ 190 % (v.s. LRU)
  - ↑2 ~ 17 % (v.s. ARC)

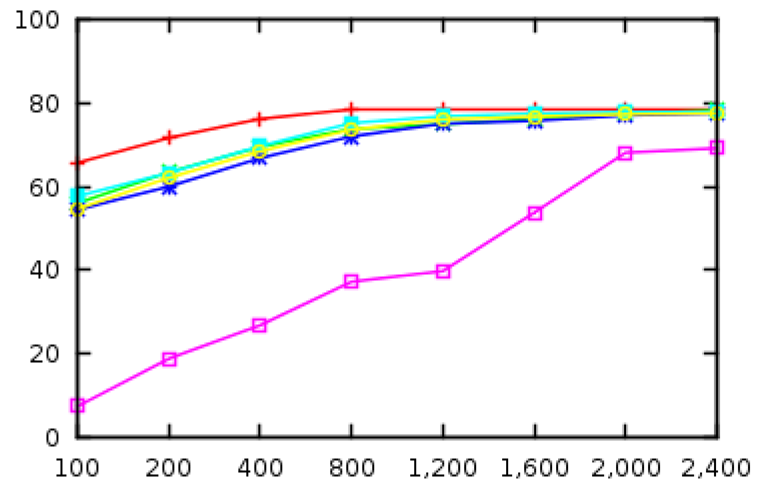


Simulation

- Hit Rate(*webvm*)
  - ↑14 ~ 22 % (v.s. LRU)

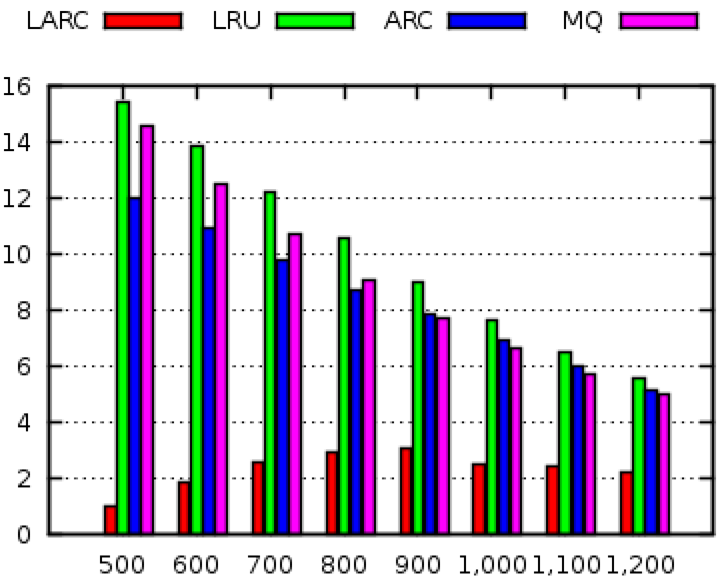


- Hit Rate(*homes*)
  - marginal improvements

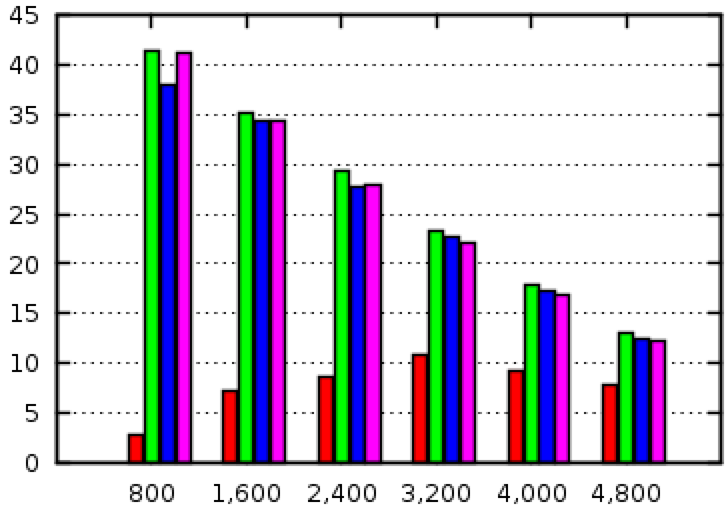


Simulation

- Cache Writes(*websearch*)
  - ↓60 ~ 94 % (v.s. LRU)

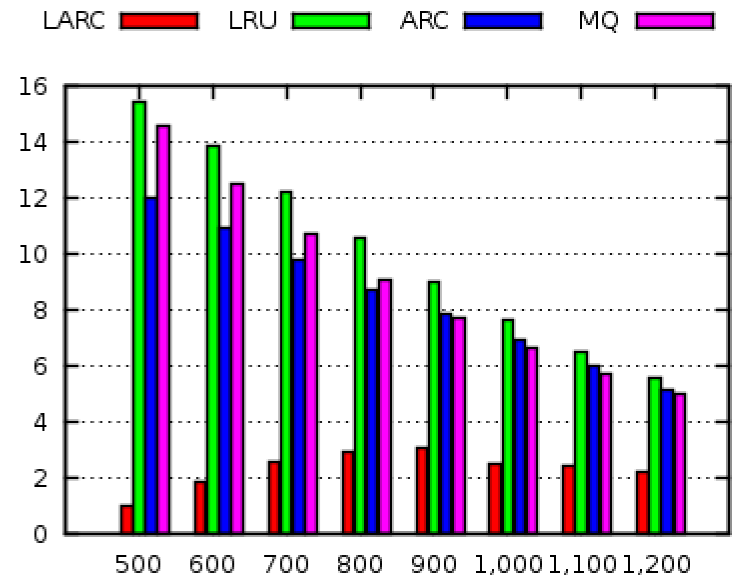


- Cache Writes(*ads*)
  - ↓40 ~ 93 % (v.s. LRU)

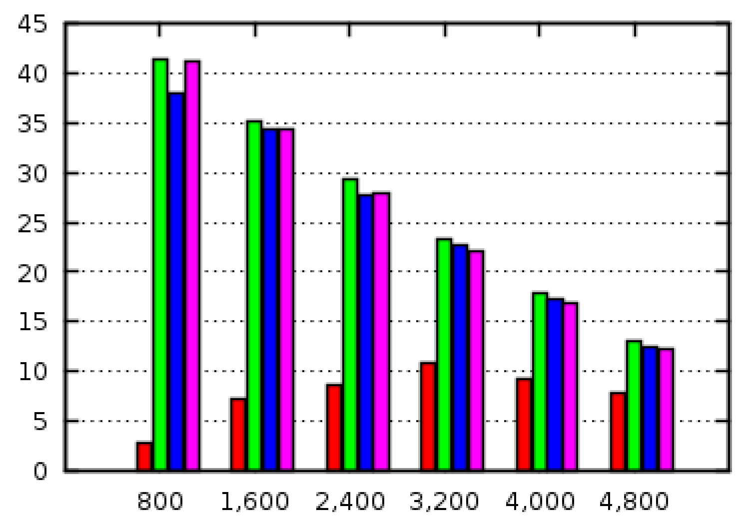


Simulation

- Cache Writes(*webvm*)
  - ↓16 ~ 25 % (v.s. LRU)

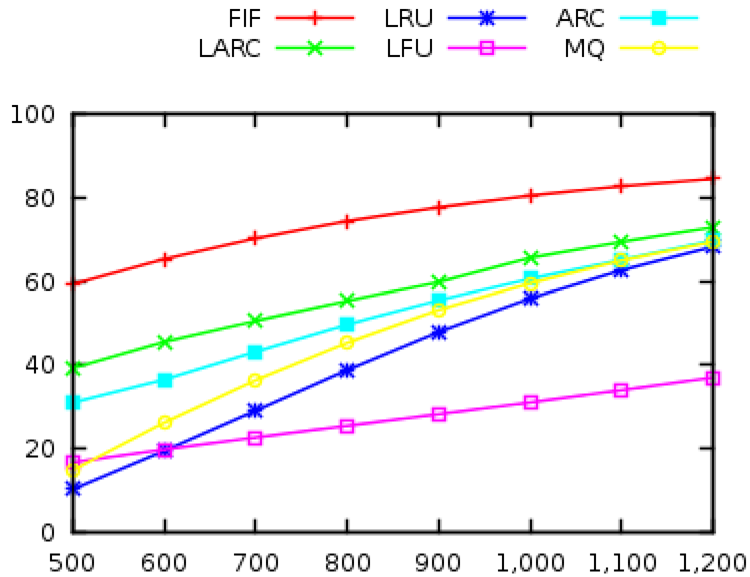


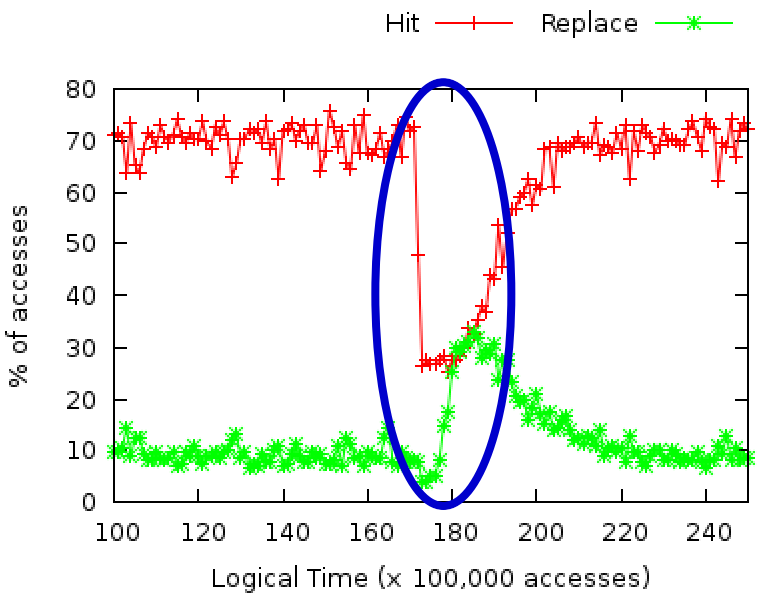
- Cache Writes(*homes*)
  - ↓11 ~ 40 % (v.s. LRU)

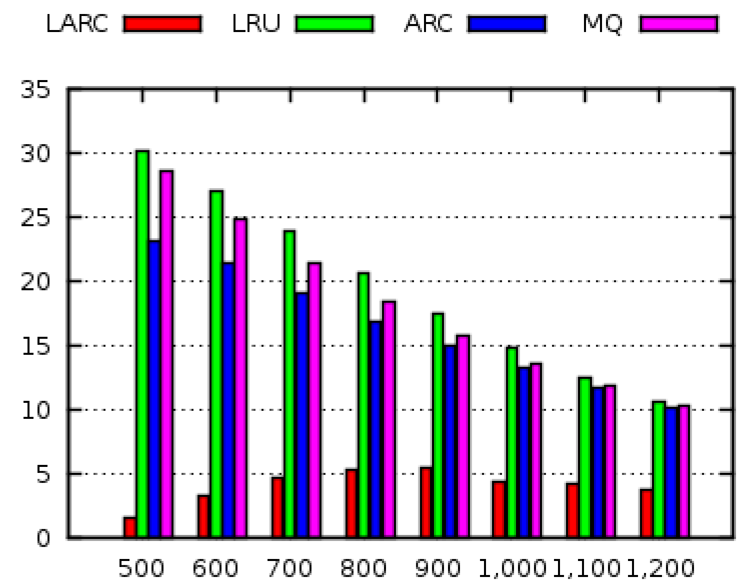


Simulation

- *ws_con*
  - similar performance to *websearch*
  - "lazy" but effective







Prototype & Evaluation

- Prototype in Flashcache
  - 200+ lines of code
- Evaluated with Filebench
  - two workloads
  - Γ ($\alpha = 0.1$, $\beta = 1.0$)
- Results
  - IOPS
  - blocks written to SSD/op
  - average of 3 runs

Testbed

DRAM	2G DDR2-667MHz
SSD	Intel SSDSA2SH064G1GC 64GB
HDD	Seagate ST373207LW 73GB
OS	Scientific Linux 6.3 2.6.32-279.5.1.el6
File System	ext4
Benchmark	Filebench-1.4.9.1

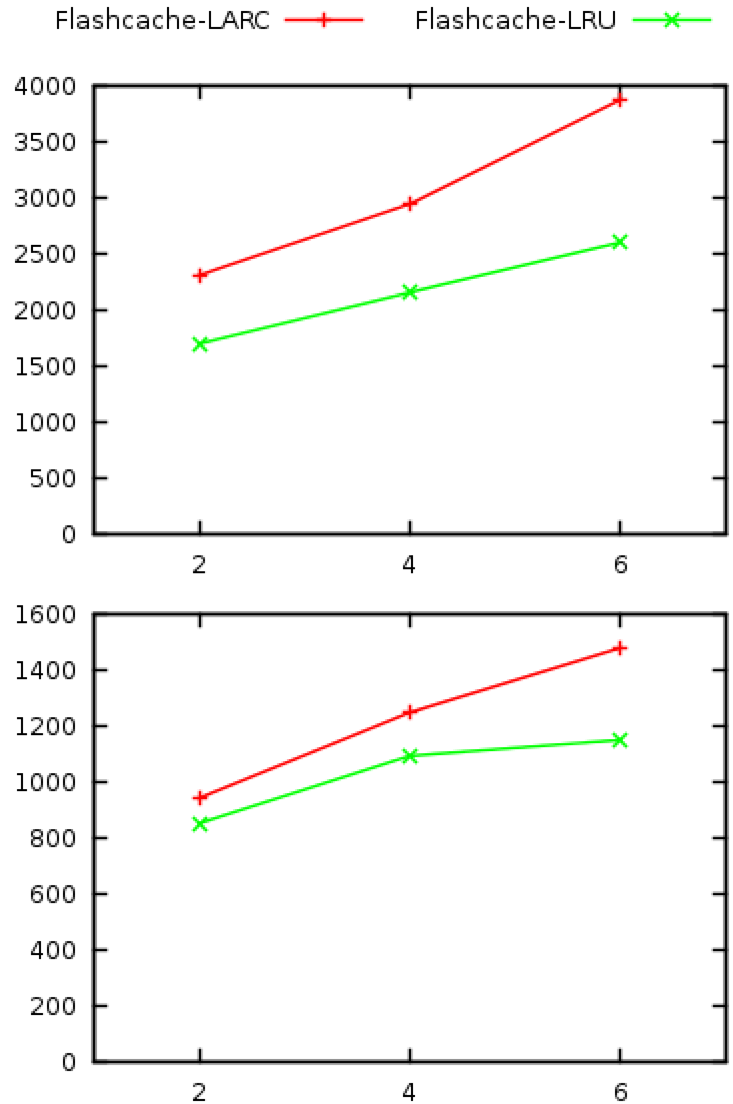
Workloads

	avg. file size	# of files	r/w
webserver	32KB	250,000	10:1
netsfs	2.4KB	500,000	1:5

Prototype & Evaluation

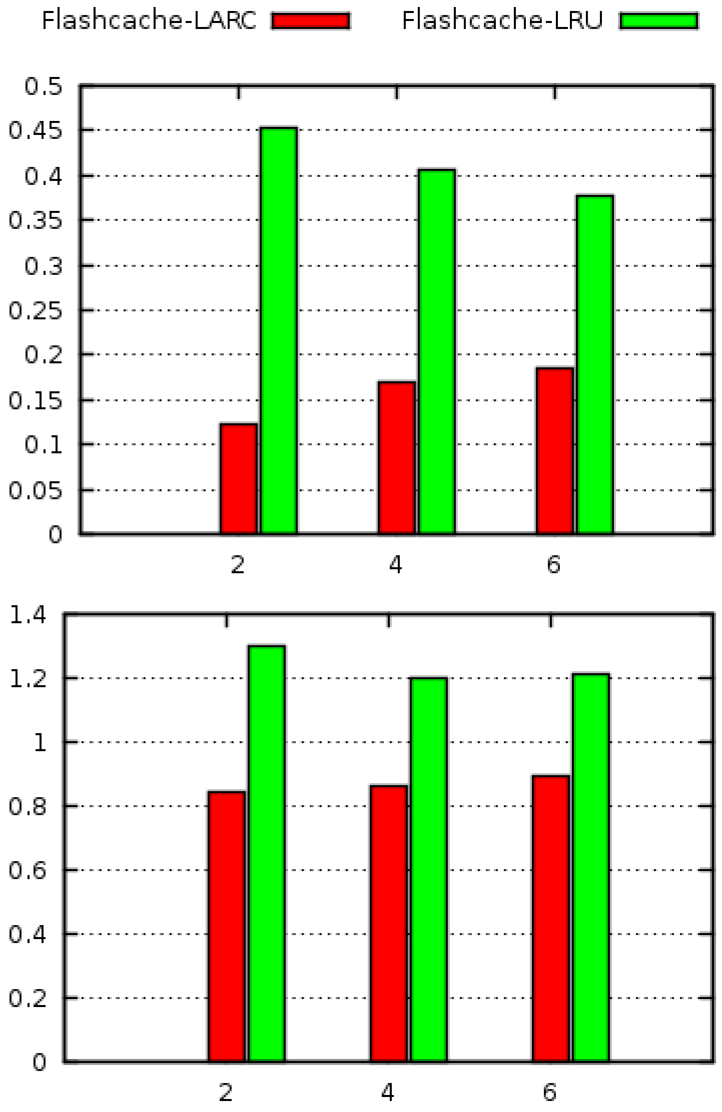
- IOPS(*webserver*)
  - ↑36 ~ 49 %

- IOPS(*netsfs*)
  - ↑10 ~ 29 %



Prototype & Evaluation

- SSD writes(*webserver*)
  - ↓51 ~ 73 %
- SSD writes(*netsfs*)
  - ↓26 ~ 35 %



Conclusion

- LARC
  - exploits the skewed popularity of blocks and keeps seldom accessed blocks out of cache
  - improves hit rate and extends SSD lifetime simultaneously
  - self-tunning and adapts to different kinds of workloads
  - low-overhead and scan-resistant
  - works better for read-only cache or read-intensive workloads

Thank you !

seth.hg@gmail.com,
dfeng@hust.edu.cn