Azor: Using Two-level Block Selection to Improve SSD-based I/O caches

Yannis Klonatos, Thanos Makatos, Manolis Marazakis,
Michail D. Flouris, Angelos Bilas

{*klonatos, makatos, maraz, flouris, bilas*}*@ics.forth.gr*

Foundation for Research and Technology - Hellas (FORTH),
Institute of Computer Science (ICS)

July 23, 2011

Table of contents

Background

- Increased need for high-performance storage I/O
  1. Larger file-set sizes ⇒ more I/O time
  2. Server virtualization and consolidation ⇒ more I/O pressure
- SSDs can mitigate I/O penalties

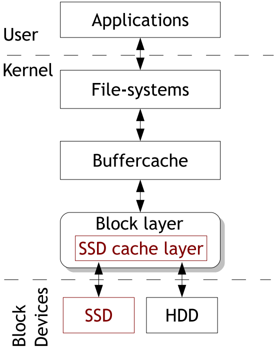
	SSD	HDD
Throughput (R/W) (MB/s)	277/202	100/90
Response time (ms)	0.17	12.6
IOPS (R/W)	30,000/3,500	150/150
Price/capacity ($/GB)	$3	$0.3
Capacity per device	32 – 120 GB	Up to 3TB

- Mixed SSD and HDD environments are necessary
- Cost-effectiveness: deploy SSDs as HDDs caches

Previous Work

- Web servers as a secondary file cache[*Kgil et al., 2006*]
    - ▷ Requires application knowledge and intervention
- *Readyboost* feature in Windows
    - ▷ Static file preloading
    - ▷ Requires user interaction
- *bcache* module in the Linux Kernel
    - ▷ Has no admission control
- *NetApp's Performance Acceleration Module*
    - ▷ Needs specialized hardware

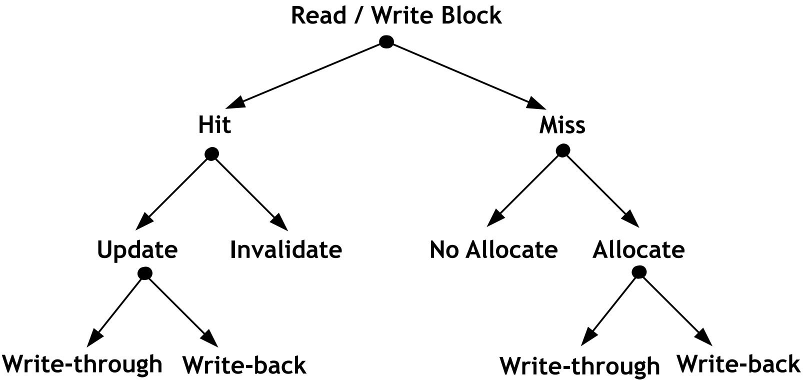
Our goal



- Design *Azor*, a transparent SSD cache
  - ▷ Move SSD caching to block-level
  - ▷ Hide the address space of SSDs
- Thorough analysis of design parameters
  1. Dynamic differentiation of blocks
  2. Cache associativity
  3. I/O concurrency

Table of contents

Overall design space

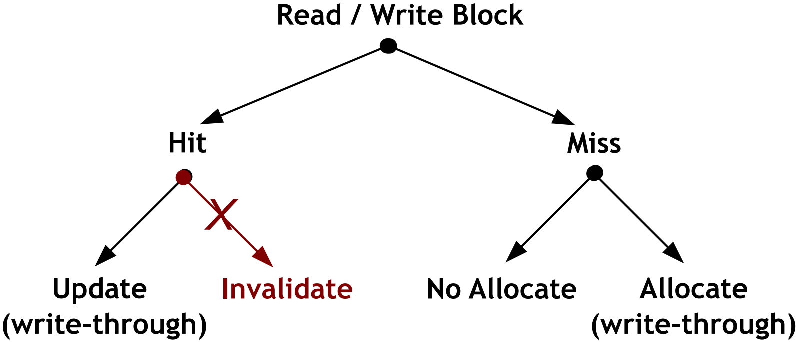


Writeback Cache Design Issues

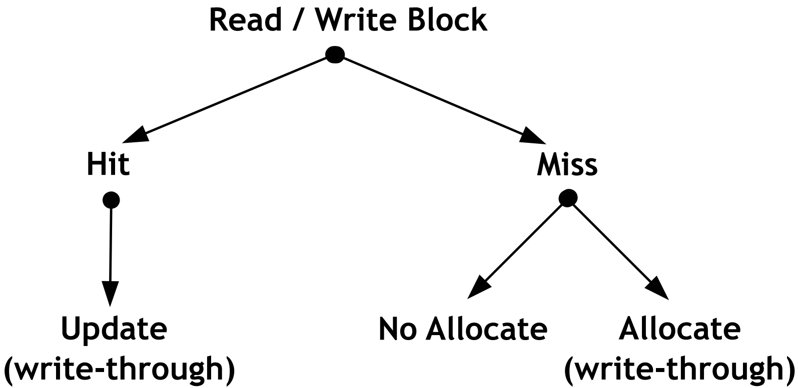
1. Requires synchronous metadata updates for write I/Os,
   - HDDs may not have the up-to-date blocks
   - Must know the location of each block in case of failure
2. Reduces system resilience to failures,
   - A failing SSD results in data loss
   - SSDs are hidden, so other layers can't handle these failures

▷ Our write-through design avoids these issues

Overall design space



Overall design space



Read / Write Block

Hit

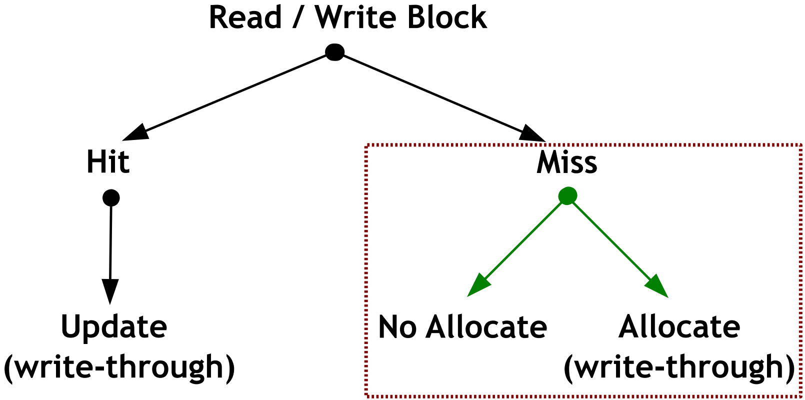
Miss

Update (write-through)

No Allocate

Allocate (write-through)

Overall design space



Read / Write Block

Hit

Update (write-through)

Miss

No Allocate

Allocate (write-through)

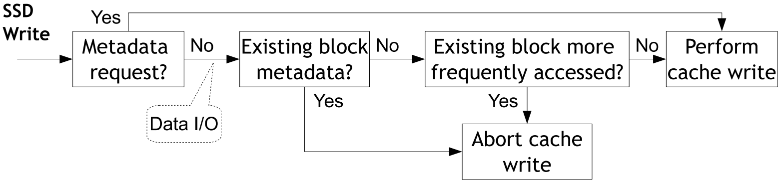
Dynamic block differentiation

- Blocks are not equally important to performance
  - ▷ Makes sense to differentiate during admission to SSD cache

- Introduce a **2**-**L**evel **B**lock **S**election scheme (2LBS)
- First level: Prioritize filesystem metadata over data
  - ▷ Many more small files → more FS metadata
  - ▷ Additional FS metadata introduced for data protection
  - ▷ Cannot rely on DRAM for effective metadata caching
  - ▷ Metadata requests represent 50% – 80% of total I/O accesses [⋆]
- Second level: Prioritize between data blocks
  - ▷ Some data are accessed more frequently
  - ▷ Some data are used for faster accesses to other data

[⋆] *D. Roselli and T. E. Anderson, "A comparison of file system workloads", Usenix ATC 2000*

Two-level Block Selection



- Modify XFS filesystem to tag FS metadata requests
  - ▷ Transparent metadata detection also possible
- Keep in DRAM an estimate of each HDD block's accesses
  - ▷ Static allocation: 256 MB DRAM required per TB of HDDs
  - ▷ DRAM space required is amortized with better performance
  - ▷ Dynamic allocation of counters reduces DRAM footprint

Cache Associativity

- Associativity: performance and metadata footprint tradeoff
- Higher-way associativities need more DRAM space for metadata

- Direct-Mapped cache
  - ▷ Minimizes metadata requirements
  - ▷ Suffers from conflict misses
- Fully-Set-Associative cache
  - ▷ 4.7× more metadata than the direct-mapped cache
  - ▷ Proper choice of replacement policy is important

Cache Associativity - Replacement policy

- Large variety of replacement algorithms used in CPUs/DRAM
  - ▷ Prohibitively expensive in terms of metadata size
  - ▷ Assume knowledge of the workload I/O patterns
  - ▷ May cause up to 40% performance variance
- We choose the LRU replacement policy
  - ▷ Good reference point for more sophisticated policies
  - ▷ Reasonable choice since buffer-cache uses LRU

I/O Concurrency

A high degree of I/O concurrency:

- Allows overlapping I/O with computation
- Effectively hides I/O latency

1. Allow concurrent read accesses on the same cache line
   - Track only pending I/O requests
   - Reader-writer locks per cache line are prohibitevely expensive
2. Hide SSD write I/Os of read misses
   - Copy the filled buffers to a new request
   - Introduces a memory copy
   - Must maintain state of pending I/Os

Table of contents

Experimental Setup

- Dual socket, quad core Intel Xeon 5400 (64-bit)
- Twelve 500GB SATA-II disks with write-through caching
- Areca 1680D-IX-12 SAS/SATA RAID storage controller
- Four 32GB Intel SLC SSDs (NAND Flash)
- HDDs and SSDs on RAID-0 setup, 64KB chunks
- Centos 5.5 OS, kernel version 2.6.18-194
- XFS filesystem
- 64GB DRAM, varied by experiment

Benchmarks

- I/O intensive workloads, between hours to days for each run

	Type	Properties	File Set	RAM	SSD Cache sizes (GB)
TPC-H	Data warehouse	Read only	28GB	4GB	7,14,28
SPECsfs	CIFS File-server	write-dominated, latency-sensitive	Up to 2TB	32GB	128
TPC-C	OLTP workload	highly-concurrent	155GB	4GB	77.5

Experimental Questions

Which is the best static decision for handling I/O misses?

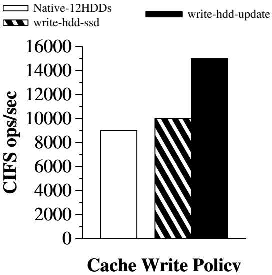
Does dynamically differentiating blocks improve performance?

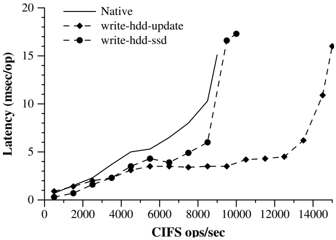
How does cache associativity impact performance?

Can our design options cope with a "black box" workload?

Table of contents

Static decision for I/O misses (SPECsfs2008)

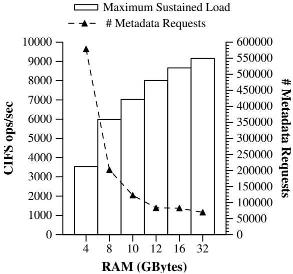


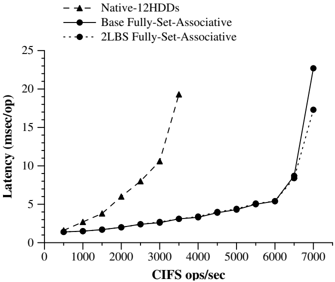


- 11% to 66% better performance than HDDs
- Huge file set, only 30% accessed
  - ▷ *write-hdd-ssd* policy evicts useful blocks
- Up to 5000 CIFS ops/sec difference for the same latency

Differentiating filesystem metadata (SPECsfs2008)

- FS metadata continuously increase during execution

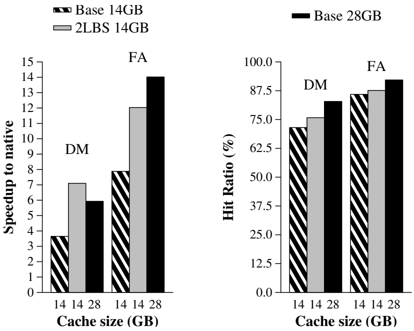




- Metadata DRAM misses ⇒ up to 71% impact
- DRAM data hit ratio less than 5%
- 3,000 more CIFS ops/sec between HDDs and *Azor*
- ∼23% latency reduction when using 2LBS in *Azor*

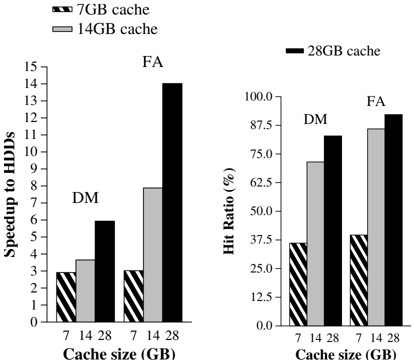
Differentiating filesystem data blocks (TPC-H)

- Filesystem data like indices important for databases
- Data differentiation improves performance



- 1.95× and 1.53× improvement for DM and FA caches
- Medium size DM is 20% better than large size DM
  - → With 10% less hit ratio

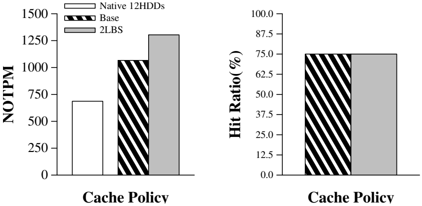
Importance of cache associativity (TPC-H)



- FA better than DM for all cache sizes
  - ▷ Large size FA = 1.36× better than DM counterpart
  - ▷ Up to 15% less conflict misses than DM
  - ▷ Medium size FA 32% better than large size DM

A black box workload (TPC-C)

- We choose the best parameters found so far
  - *Fully-set-associative* cache design
  - SSD cache size of half the workload size



- Base cache: 55% improvement to native
- 2LBS cache: 34% additional improvement
- Hit ratio remains the same in both versions
- Disk utilization is 100%, SSD utilization under 7%

Table of contents

Conclusions

- We use SSD-based I/O caches to increase storage performance
- Performance is improved with higher way associativities
  - At the cost of 4.7× higher metadata footprint
- Write policy can make up to 50% performance difference
- We explore differentiation of HDD blocks
  - According to their expected importance on system performance
  - Design and evaluation of a two-level block selection scheme
- Overall, our work shows that differentiation of blocks is a promising technique for improving SSD-based I/O caches