Mon. December 23rd, 2013
@High Performance Computing Class

Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems

25th International Conference on Supercomputing
(ICS2011), May 31 - June 4, 2011
Sponsored by ACM/SIGARCH

Author: Feng Chen† David Koufaty† Xiaodong Zhang††
† Circuits and Systems Research Intel Labs
†† Dept. of Computer Science & Engineering
The Ohio State University

Presenter: Ryohei Kobayashi (13D38025)

The Selected Paper

- ICS: one of the top conference

Microsoft Academic Search Beta

Advanced Search

Authors »
Publications »
Conferences »
Journals »
Keywords »
Organizations »

Academic > Computer Science > Distributed & Parallel Computing

Top conferences in distributed & parallel computing

1–100 of 203 results

All Years

1 2 3

Conferences	Publications	Field Rating
ICDCS - International Conference on Distributed Computing Systems	2364	86
PODC - Symposium on Principles of Distributed Computing	1520	83
SC - Supercomputing Conference	3066	81
IPDPS(IPPS) - International Parallel and Distributed Processing Symposium/International Parallel Processing Symposium	5462	61
5 ICS - International Conference on Supercomputing	1180	61
HPDC - IEEE International Symposium on High Performance Distributed Computing	1010	61
FTCS - Symposium on Fault-Tolerant Computing	911	61
ICPP - International Conference on Parallel Processing	3247	59
SPAA - ACM Symposium on Parallel Algorithms and Architectures	931	59
NSDI - Networked Systems Design and Implementation	288	58
PACT - International Conference on Parallel Architectures and Compilation Techniques	802	48

- Best Paper Award

``Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems", *Proceedings of 25th ACM International Conference on Supercomputing* (ICS 2011), Tucson, Arizona, May 31 - June 4, 2011. **Best Paper Award** .

Abstract

- This paper shows how to make the best use of SSD in storage systems with insights based on the design and implementation of a high performance hybrid storage system, called *Hystor*.
  - SSD should play a major role as an independent storage where the best suitable data are adaptively and timely migrated in and retained.
  - It can also be effective to serve as a write-back buffer.

Abstract - What is the Hystor? -

- Hystor: A Hybrid Storage System
- Hystor manages both SSDs and HDDs as one single block device with minimal changes to existing OS kernels.
- Monitoring I/O access patterns at runtime
  - Hystor can effectively identify following blocks and store them in SSD
    - (1)Blocks that can result in long latencies
    - (2)Blocks that are semantically critical (e.g. file system metadata)
- In order to further leverage the exceptionally high performance of writes in the state-of-the-art SSD, SSD is also used as write-back buffer
  - To speed up write requests
- This Study on Hystor implemented in the Linux kernel 2.6.25.8 shows
  - it can take advantage of the performance merits of SSDs with only a few lines of changes to the stock Linux kernel.

Agenda

- Introduction
- SSD Performance Advantages
- High-Cost Data Blocks
- Maintaining Data Access History
- The Design and Implementation of Hystor
- Evaluation
- Conclusion and Impression

Agenda

- Introduction ☜
- SSD Performance Advantages
- High-Cost Data Blocks
- Maintaining Data Access History
- The Design and Implementation of Hystor
- Evaluation
- Conclusion and Impression

Introduction

- SSDs are becoming an important part of high-performance storage systems

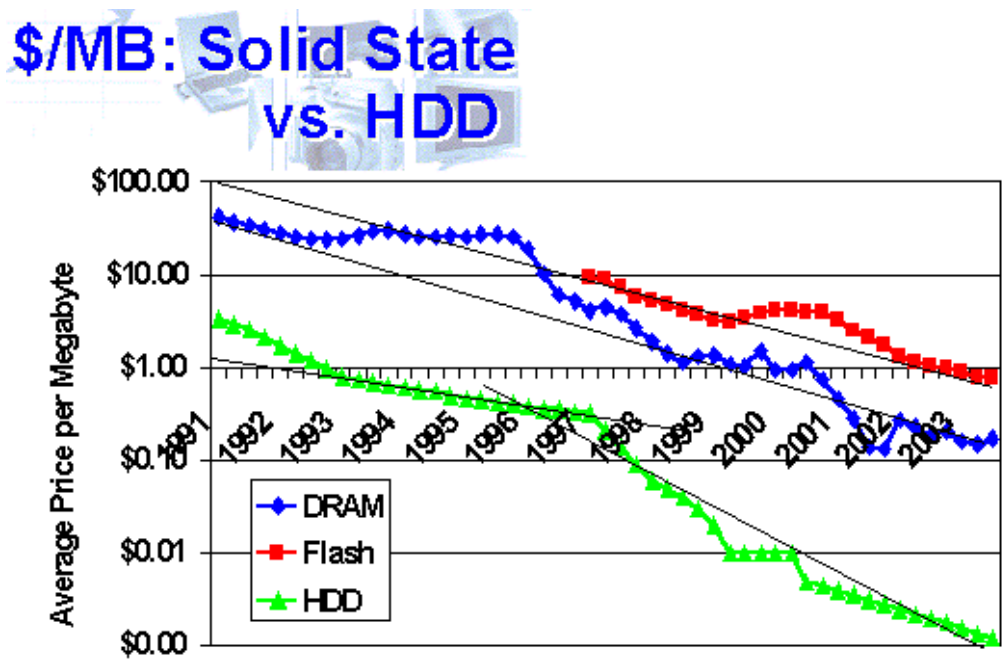
'Gordon' Supercomputer
@San Diego Supercomputer Center(SDSC)

- A flash-based supercomputer [ASPLOS'09]
- Adopting 256TB of flash memory as storage*
- $20 million funding from the National Science Foundation (NSF)

* http://www.internetnews.com/hardware/article.php/3847456

Introduction

- SSD's disadvantages
  - Relatively high price and low capacity
    - E.g. around $12/GB (32GB Intel® X25-E SSD)
      - 100 times more expensive than a typical commodity HDD



http://www.storagesearch.com/semico-art1.html

Introduction

- It's Unsuitable to built a storage system completely based on SSDs
  - Especially, for most commercial and daily operated systems

Thus…

- Authors believe that SSDs should be a means to enhance the existing HDD-based storage
  - Only by finding the fittest position in storage systems, it's possible to strike a right balance between *performance* and *cost*

Introduction

- Contributions of this work
  - Identifying an effective metric to represent the performance-critical blocks by considering both temporal locality and data access patterns

  - Design of an efficient mechanism to profile and maintain detailed data access history for a long-term optimization

  - A comprehensive design and implementation of a high performance hybrid storage system
    - improving performance for accesses to the high-cost data blocks, semantically-critical (file system metadata) blocks, and write-intensive workloads with minimal changes to existing systems

Agenda

- Introduction
- SSD Performance Advantages ☜
- High-Cost Data Blocks
- Maintaining Data Access History
- The Design and Implementation of Hystor
- Evaluation
- Conclusion and Impression

SSD Performance Advantages

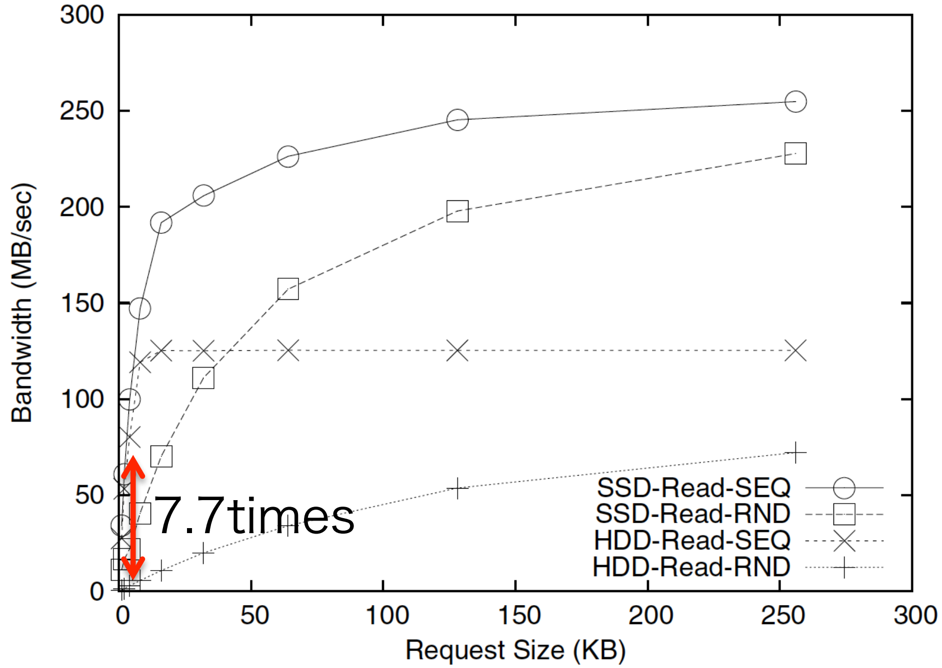
- SSD vs. HDD

	Intel® X25-E SSD	Seagate® Cheetah® HDD
Capacity	32GB	73GB
Interface	SATA2 (3.0Gb/s)	LSI® MegaRaid® 8704 SAS card
Read Bandwidth	250MB/sec	125MB/sec
Write Bandwidth	180MB/sec	125MB/sec

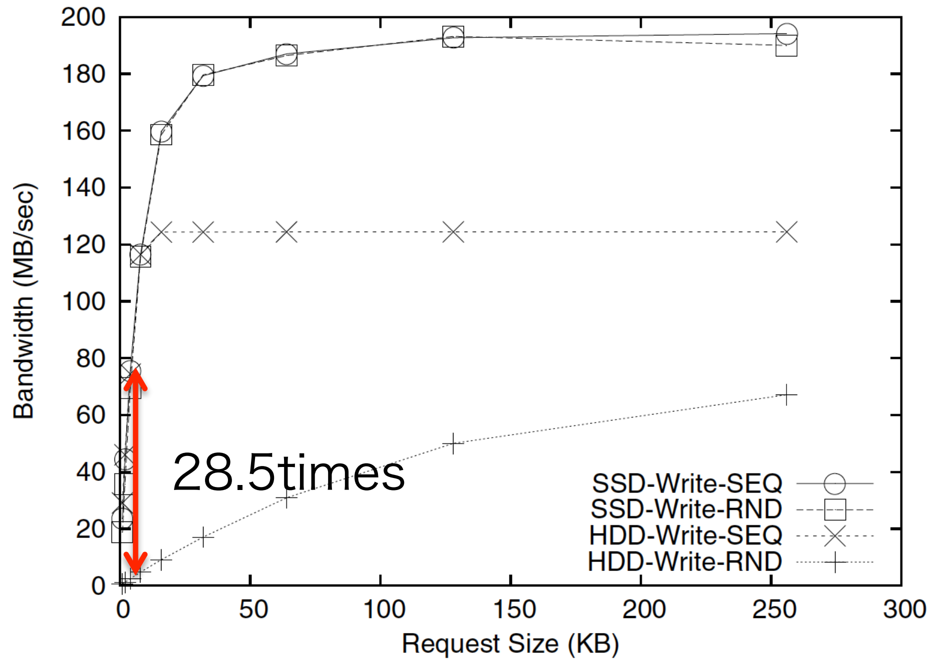
SSD Performance Advantages

*M. P. Mesnier. Intel open storage toolkit. http://www.sourceforge.org/projects/intel-iscsi

- Intel® Open Storage Toolkit*
  - generates four typical workloads: Random Read/Write, Sequential Read/Write)



(a) Reads



(b) Writes

RND Read/Write: 7.7 times and 28.5 higher bandwidths than on the HDD (Request size 4KB)

SSD Performance Advantages

- This experimental result shows
  - Achievable performance benefits are highly access patterns
  - exceptionally high write performance on the SSD (up to 194MB/sec)
  - Random write can achieve almost identical performance as sequential write
  - Writes on the SSD can quickly reach a rather high bandwidth (around 180MB/sec) with a relatively small request size (32KB) for both random and sequential workloads

SSD Performance Advantages

- Two key issues that must be considered in the design of Hystor, based on these observations
  - Need to recognize workload access patterns to identify the most high-cost data blocks, especially those blocks being randomly accessed by small requests
    - It cause the worst performance for HDDs.
  - To leverage the SSD as a write-back buffer to handle writes
    - Need not to treat random writes specifically, since random/ sequential write performance are almost same

Agenda

- Introduction
- SSD Performance Advantages
- High-Cost Data Blocks ☜
- Maintaining Data Access History
- The Design and Implementation of Hystor
- Evaluation
- Conclusion and Impression

High-Cost Data Blocks

- Many workloads have a *small* data set
  - Contributing a large percentage of the aggregate latency in data access

Thus…

- Hystor's critical task is
  - to identify the most performance-critical blocks

Identifying high-cost blocks

- Prior work - *Experiences in building a software-based SATF scheduler* [*Tech. Rep. ECSL-TR81*, 2001.] -
  - Maintaining an on-line hard disk model to predict the latency for each incoming request
  - Heavily depending on precise hard disk modeling based on detailed specification data
    - it is often unavailable in practice
  - As the HDD internals become more complicated(e.g. disk cache), it's difficult
    - to accurately model a modern hard disk
    - to precisely predict the I/O latency for each disk access

Identifying high-cost blocks

- Author's approach
  - Using a pattern-related metric as *indicator* to indirectly *infer*(≒ estimate) access cost without need of knowing the exact latencies
  - Associating each data block with a selected metric and update the metric value by observing access to the data block

- The approach's key issue
  - Selected metric should have a strong correlation to access latency
    - to effectively estimate the *relative* access latencies associated to blocks
    - to identify the relatively high-cost data blocks

Indicator Metrics

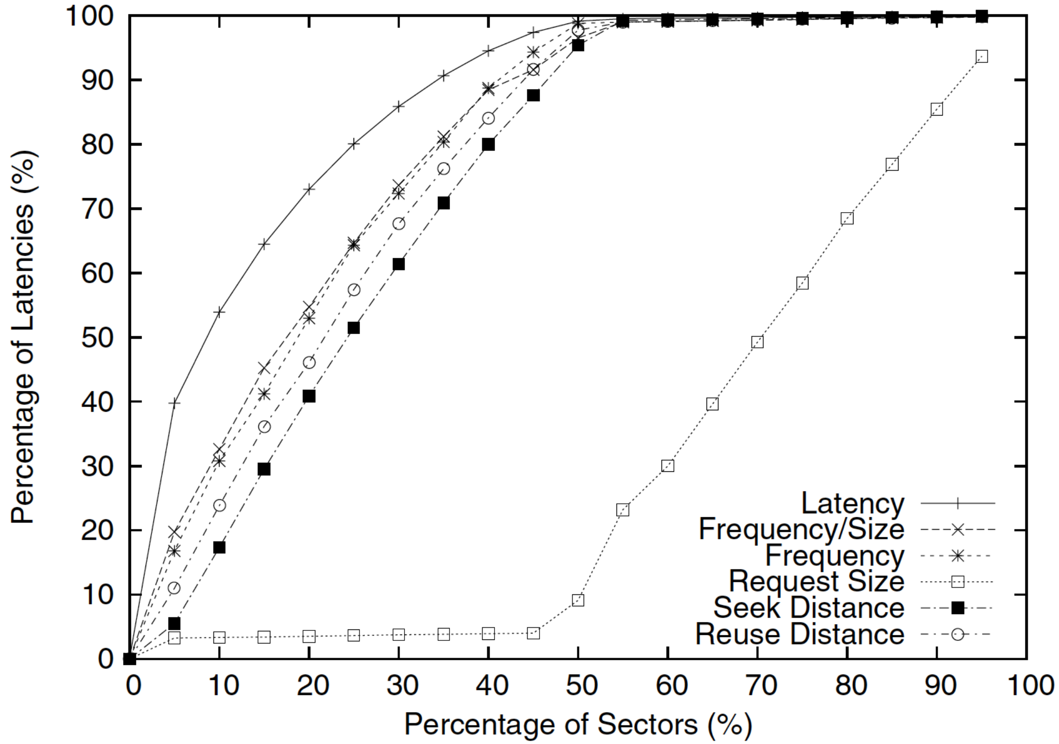
- Four candidates (Note considering their combinations)
  - Request size
  - Frequency
  - Seek distance
  - Reuse distance

- In order to evaluate how highly these candidates are correlated to access latencies, *blktrace** tool is used
  - This tool collects I/O traces on an HDD for a variety of workloads

*Blktrace. http://linux.die.net/man/8/blktrace.

Indicator Metrics

- Accumulated HDD latency of sectors sorted in descending order by using candidate metrics in TPC-H workload
- The closer a curve is to the *latency* curve, the better the corresponding metric is
  - Frequency/Request Size is most effective one



Indicator Metrics – Frequency/Request Size -

- Frequency: Temporal locality
  - This metric is used to avoid the *cache pollution problem* for handling weak-locality workload [USENIX'05]
- Request Size: Access pattern
  - The average access latency per block is highly correlated to request size
    - Because a large request can effectively amortize the seek and rotational latency over many blocks
  - The request size also reflects workload access patterns
    - the sequence of data accesses observed at the block device level is an optimized result of multiple upper-level components (e.g. the I/O scheduler attempts to merge consecutive small requests into a large one)
  - Small requests also tend to incur high latency
    - Because they are more likely to be intervened by other requests

Frequency/Request Size metric performs consistently the best in various workloads and works well

Agenda

- Introduction
- SSD Performance Advantages
- High-Cost Data Blocks
- Maintaining Data Access History ☜
- The Design and Implementation of Hystor
- Evaluation
- Conclusion and Impression

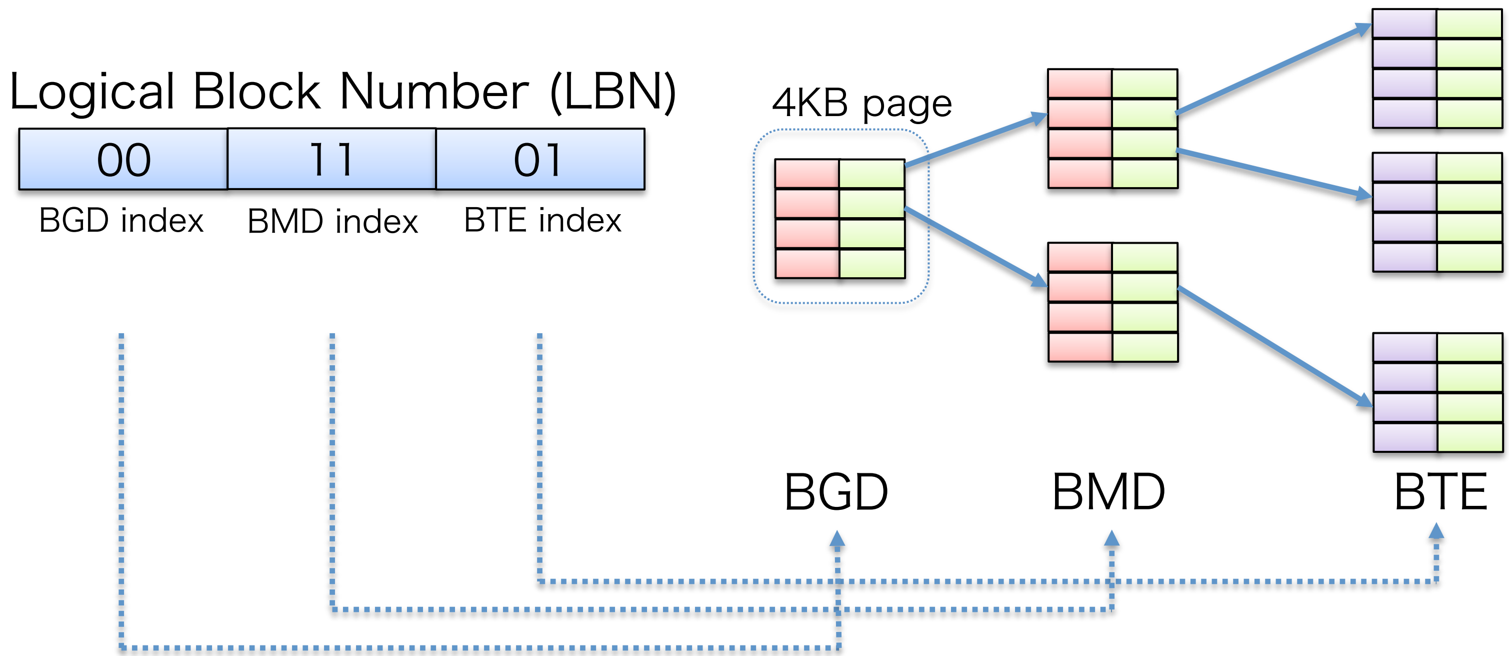
Maintaining Access History

- To use the metric values to profile data access history, two critical challenges must be addressed
  - How to represent the metric values in a compact and efficient way
  - How to maintain such history information for each block of a large-scale storage space (e.g. Terabytes)

Author's Approach

- The Block Table [FAST'05]
  - Similar to the page table used in virtual memory management
  - It has three levels
    - Block Global Directory (BGD)
      - represents the storage space segmented in units of *regions*
    - Block Middle Directory (BMD)
      - represents the storage space segmented in units of *sub-regions*
    - Block Table Entry (BTE)
      - represents the storage space segmented in units of *blocks*

The Block Table



	Name	Feature
	Unique field (16-bit)	Tracking the # of BTE entries belonging to data access information
	Counter field (16-bit)	Recording data access information
	Flag field (16-bit)	Recording other properties of a block (e.g. Whether a block is a metadata block)

Representing Indicator Metric

- Inverse bitmap
  - A technique to encode the *request size* and *frequency* in the block table
  - When a block is accessed by a request of *N* sectors, an *invers bitmap (b)* is calculated using the following equation:

$$b = 2^{max(0,7-\lfloor log_2 N \rfloor)}$$

Representing Indicator Metric

- Inverse bitmap (*b*)
  - representing the size for a given request

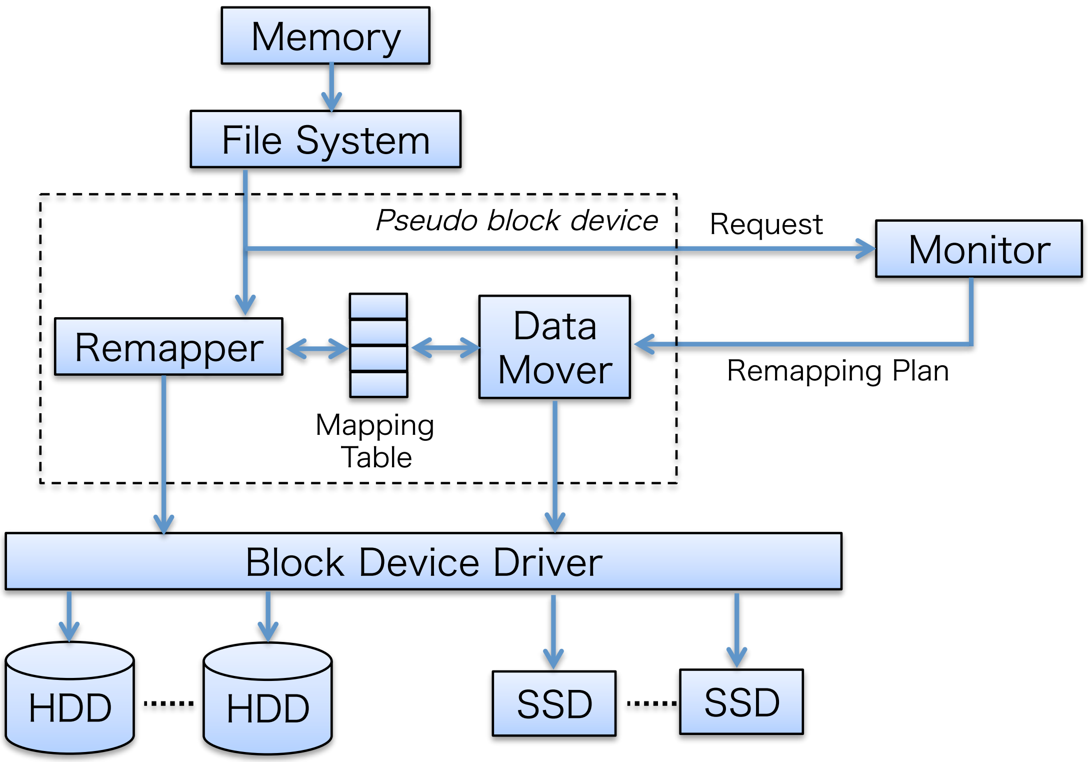
- Counter value of each entry at each level of the block table
  - representing the indicator metric *frequency/ request size*
    - Upon an incoming request, the counter of the corresponding entry is incremented by *b*

Agenda

- Introduction
- SSD Performance Advantages
- High-Cost Data Blocks
- Maintaining Data Access History
- The Design and Implementation of Hystor ☜
- Evaluation
- Conclusion and Impression

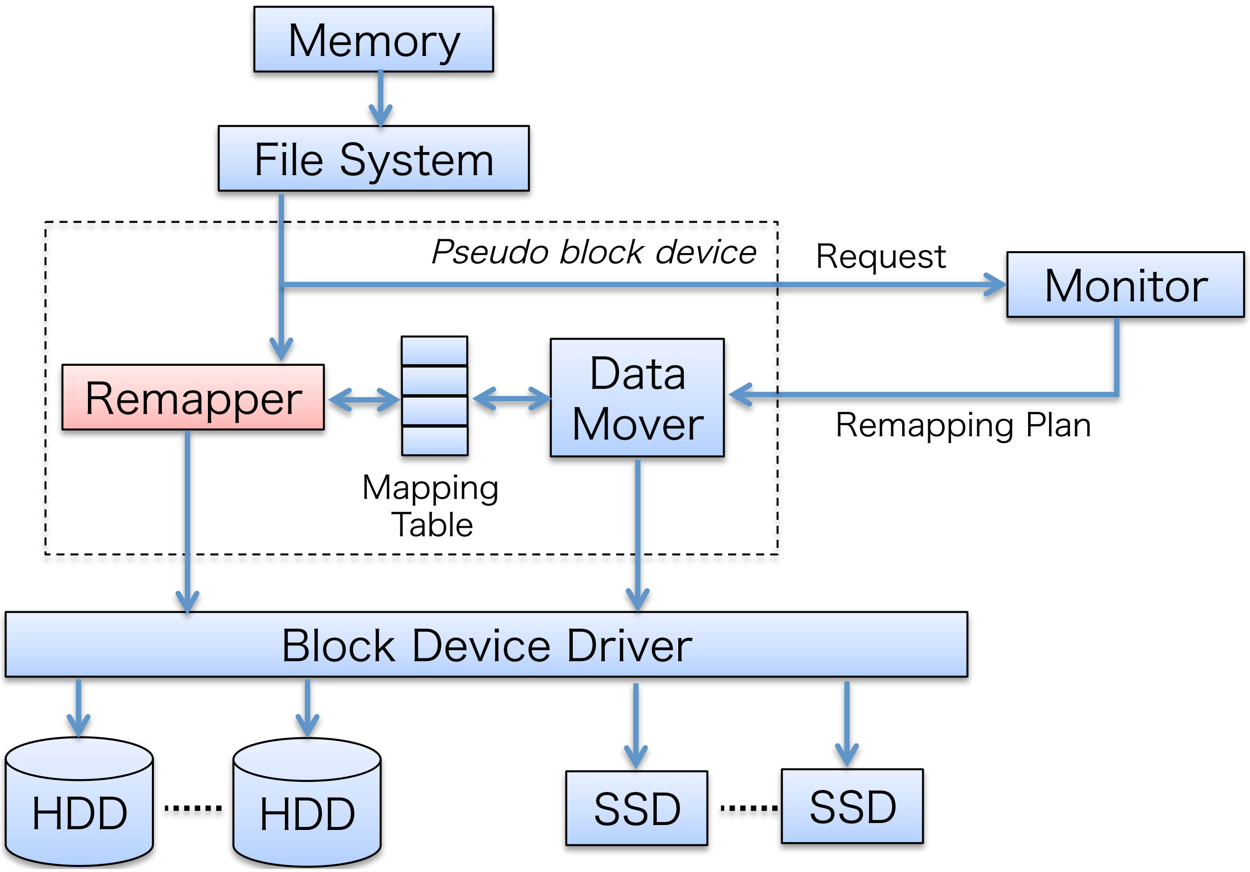
The Design of Hystor

- Main Architecture
  - Three Major components: Remapper, Monitor, and Data mover



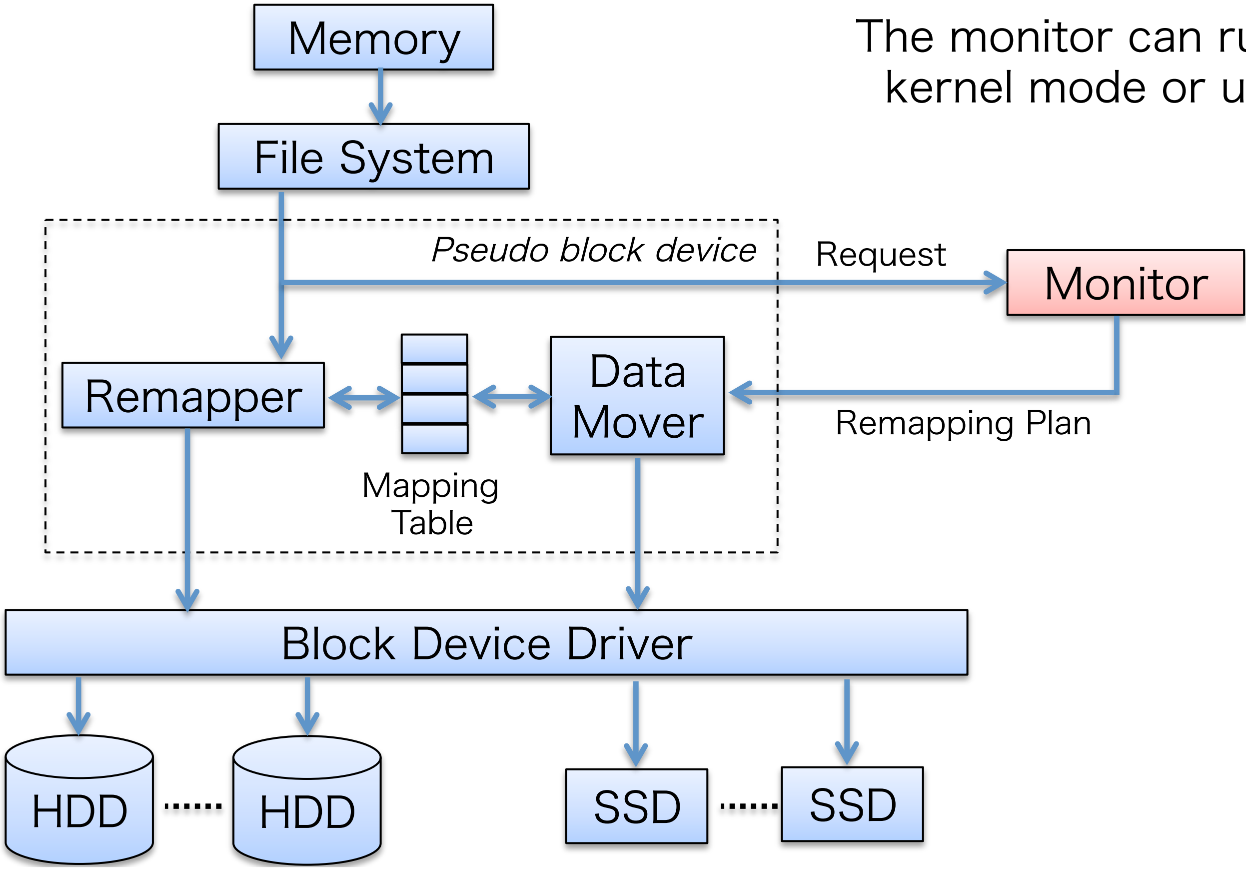
Main Architecture

- Remapper: maintaining a mapping table to track the original location of blocks on the SSD



Main Architecture

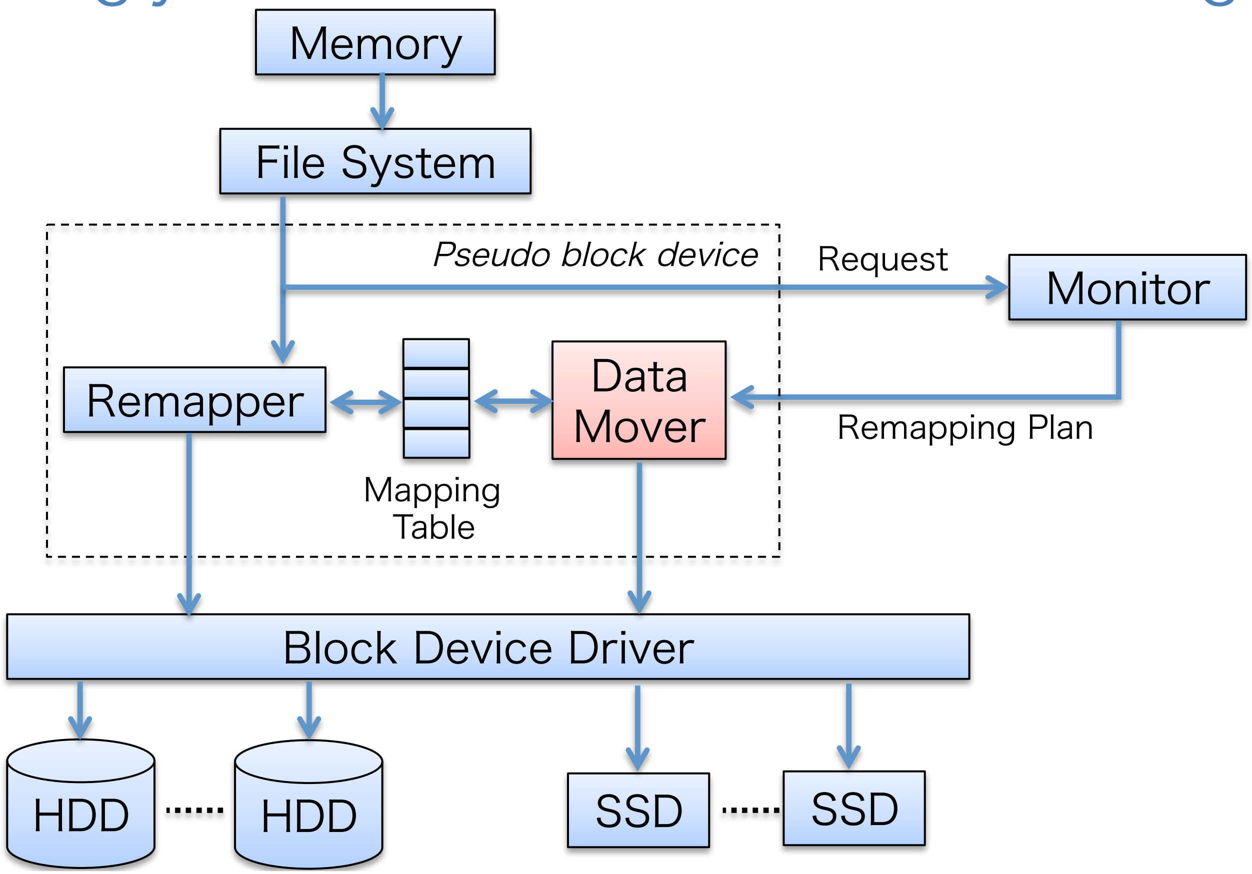
- Monitor: collecting I/O requests and updates the block table to profile workload access patterns



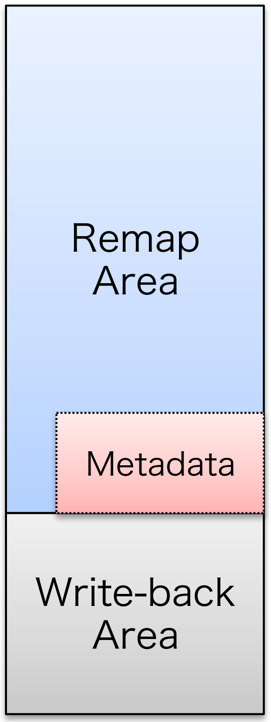
The monitor can run in either kernel mode or user mode

Main Architecture

- Data mover: issuing I/O commands to the block devices and updating the mapping table accordingly to reflect the most recent changes



SSD Space Management



SSD Space

- Remap area
  - maintaining the identified critical blocks, such as the high-cost data blocks and file system metadata blocks
  - All requests, including both reads and writes, to the blocks in the remap area are directed to the SSD
- Write-back area
  - a buffer to temporarily hold dirty data of incoming write requests
  - All other requests are directed to the HDD
  - Blocks in the write-back area are periodically synchronized to the HDD and recycled for serving incoming writes

Managing the Remap Area

- Two types of blocks can be remapped to the SSD
  - the high-cost data blocks
    - they are identified by analyzing data access history using the block table
  - file system metadata blocks
    - they are identified through available semantic information in OS kernels

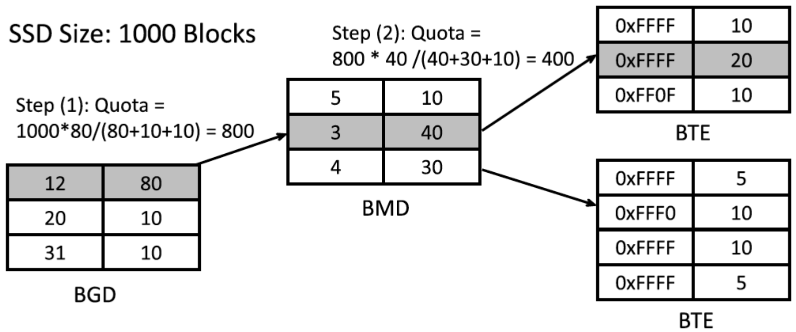
A pseudo code of identifying candidate blocks(high-cost blocks)

```
counter():   // the counter value of an entry
total_cnt(): // the aggregate value of counters of a block table page

sort_unique_asc():    // sort entries by unique values
sort_counter_dsc():  // sort entries by counter values
quota:                      // the num. of available SSD blocks
sort_unique_asc(bgd_page);  /* sort bgd entries */
bgd_count = total_cnt(bgd_page);
for each bgd entry && quota > 0; do
    bmd_quota = quota*counter(bgd)/bgd_count;  /* get the bmd page */
    bgd_count -= counter(bgd);
    quota -= bmd_quota;

    bmd_page = bgd->bmd;
    sort_unique_asc(bmd_page);  /* sort bmd entries */
    bmd_count = total_cnt(bmd_page);
    for each bmd entry && bmd_quota > 0; do
        bte_quota = bmd_quota*counter(bmd)/bmd_count;
        bmd_count -= counter(bmd);
        bmd_quota -= bte_quota;

        bte_page = bmd->bte;
        sort_counter_dsc(bte_page);
        for each bte entry && bte_quota > 0; do
            add bte to the update(candidate) list;
            bte_quota --;
        done
        bmd_quota += bte_quota;  /* unused quota */
    done
    quota += bmd_quota; /* unused quota */
done
```



- Recursively determination of the hottest blocks in the region
- Allocate SSD space to the regions correspondingly

Identifying Metadata Blocks

- A conservative approach to leverage the information that is already available in the existing OS kernels.
  - To modify a single line at the block layer to leverage this available information by tagging incoming requests for metadata blocks
  - Need not to change to file systems or applications
  - When the remapper receives a request,
    - the incoming request's tags are checked
    - the requested blocks are marked in the block table (<u>using the *flag* field of BTE entries</u>)

Managing the Write-back Area

- The blocks in the write-back area are managed in two lists
  - *clean list*
  - *dirty list*

- When a write request arrives,
  - ① SSD blocks are allocated from *clean list*
  - ② The new dirty blocks are written into the SSD and added onto the *dirty list*
  - ③ If the # of dirty blocks in the write-back area reaches a *high watermark*, these block are written-back to the HDD until reaching a *low water-mark*
    - There is a counter to track the # of dirty blocks in the write-back area
  - ④ Cleaned blocks are placed onto the clean list for reuse

Implementation

- Hystor is prototyped with about 2,500 Lines of code
  - In the Linux kernel 2.6.25.8 as a stand-alone kernel module
- Remapper
  - Based on the software RAID
- Monitor (No need any modifications in the Linux kernel)
  - User-mode
    - implemented as a user-level daemon thread with about 2,400 lines of code
  - Kernel-mode
    - It consists of 4,800 lines of code
- Kernel Changes
  - only about 50 lines of code are inserted in the stock Linux kernel

Agenda

- Introduction
- SSD Performance Advantages
- High-Cost Data Blocks
- Maintaining Data Access History
- The Design and Implementation of Hystor
- Evaluation ☜
- Conclusion and Impression

Evaluation

- Experimental System

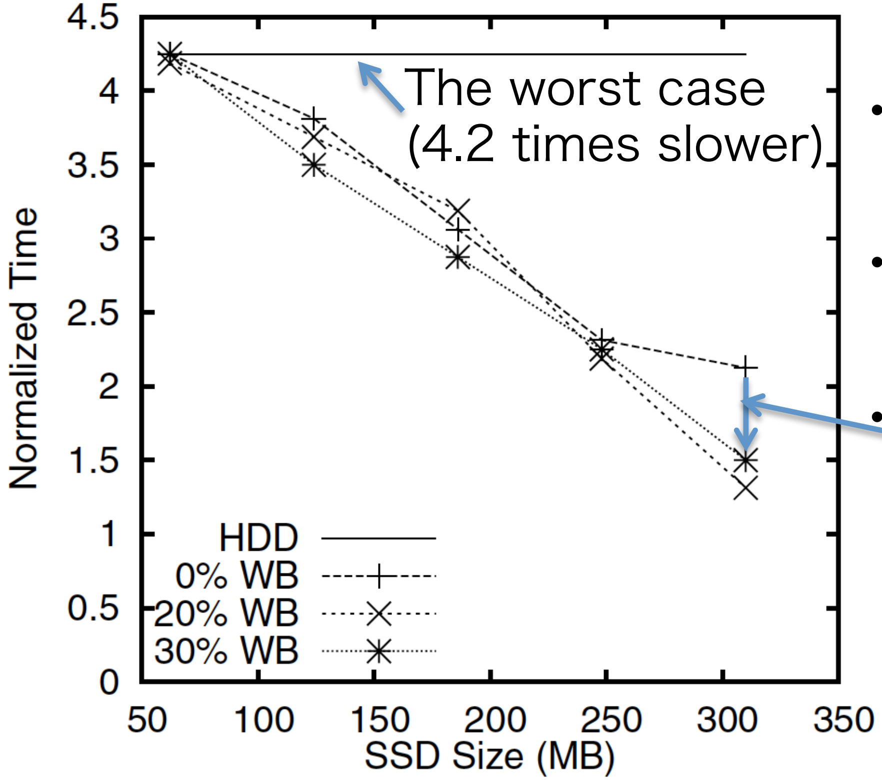
CPU	2.66GHz Intel® Core™ 2 Quad
Main Memory	4GB
Mother Board	Intel® D975BX

	Intel® X25-E SSD	Seagate® Cheetah® HDD
Capacity	32GB	73GB
Interface	SATA2 (3.0Gb/s)	LSI® MegaRaid® 8704 SAS card
Read Bandwidth	250MB/sec	125MB/sec
Write Bandwidth	180MB/sec	125MB/sec

OS	Fedora™ Core 8 with the Linux kernel 2.6.25.8
File System	Ext3 (default configuration)
Linux I/O scheduler	*No-op* (for SSDs), *CFQ* (for HDDs)
On-device Caches	Enable (all the storage devices)
The Other Configurations	Default Values

Evaluation - execution time -

- Benchmark: *Postmark**
  - small random data accesses-intensive

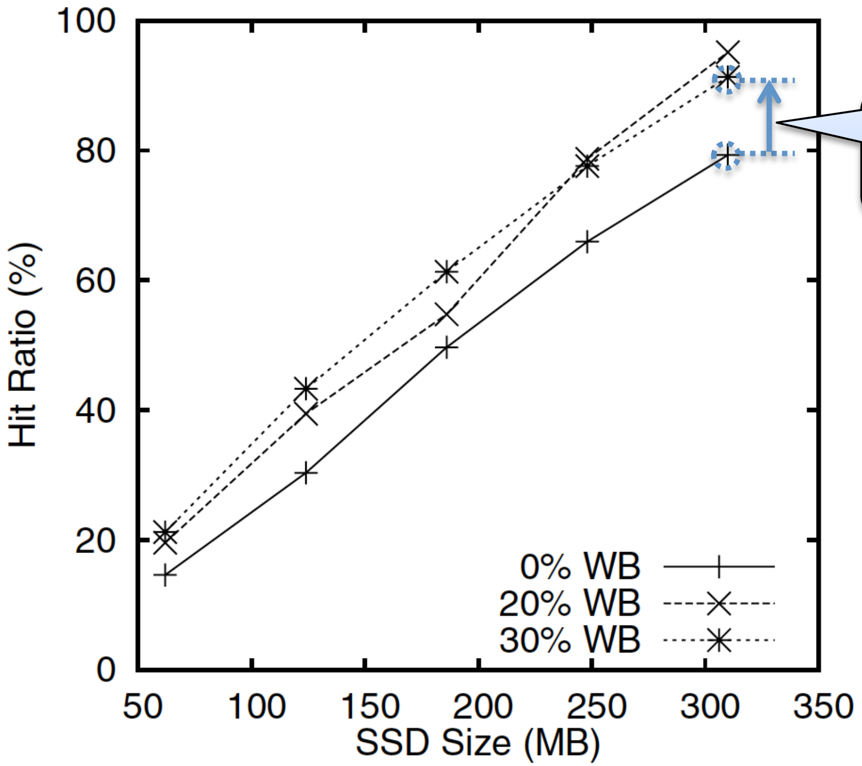


- SSD Size: 20%, 40%, 60%, 80%, and 100% of the working-set size (X-axis)
- Normalizing to execution time of running on the SSD-only system (Y-axis)
- <u>29%</u> reduction (SSD size 310MB)

*Postmark. A new file system benchmark (1997).
http://www.netapp.com/tech_library/3022.html

Evaluation - hit ratio -

- Benchmark: *Postmark*
- Y-axis: Hit ratio of I/O requests observed at the remapper (hit: A request to blocks resident in the SSD)

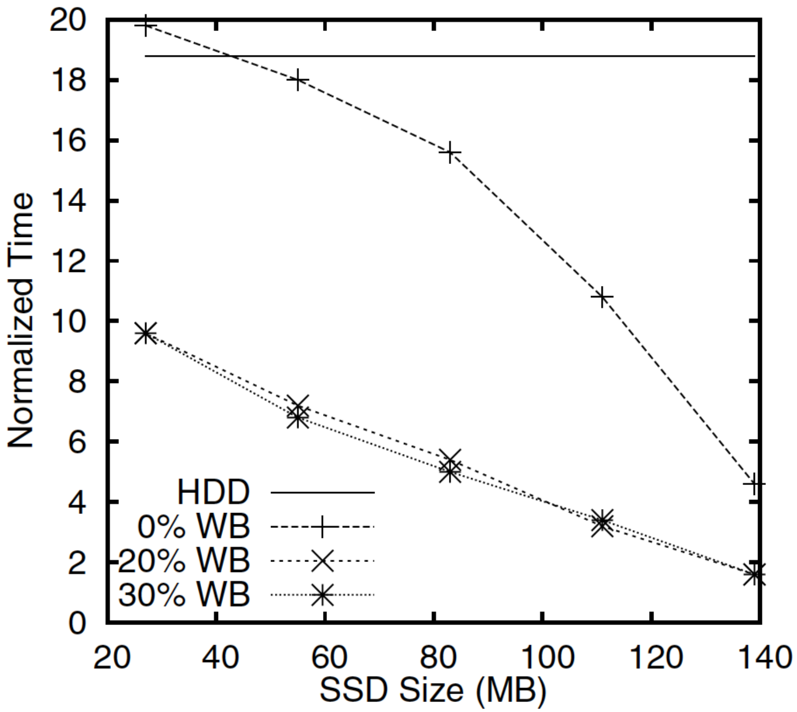


- Hit ratio is improved from 79% to 91% (SSD size 310MB)

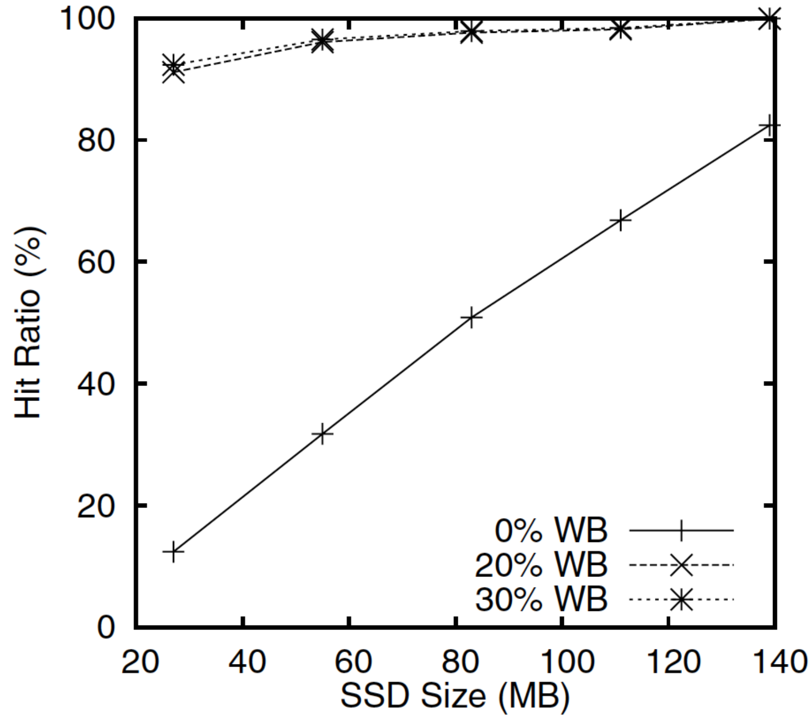
Evaluation

* S. Shah and B. D. Noble. A study of e-mail patterns. In *Software Practice and Experience*, volume 37(14), 2007.

- Benchmark: *Email**
  - intensive synchronous writes with different append sizes and locations based on realistic mail distribution function
  - a more skewed distribution of latencies
  - Most data accesses are small random writes



Execution time

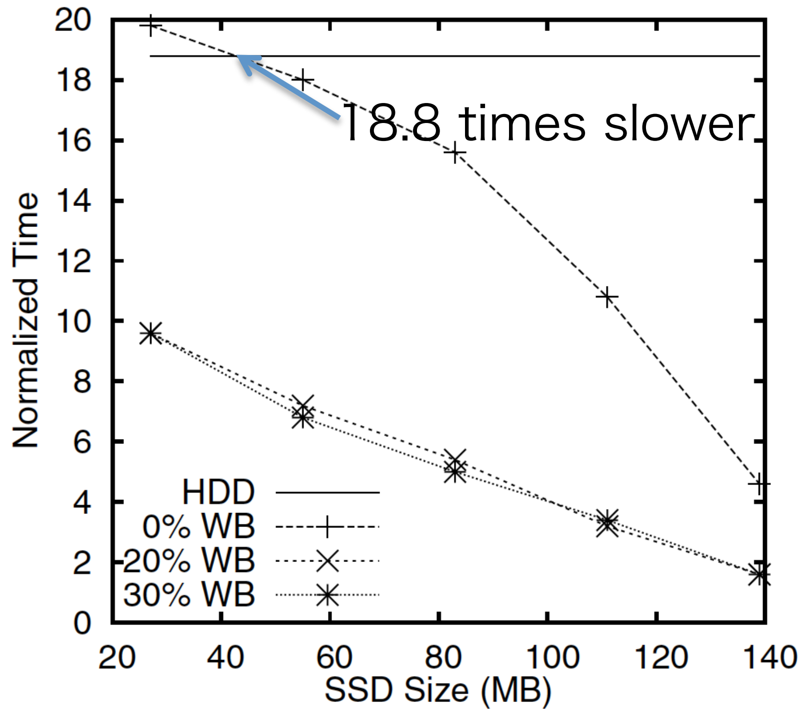


Hit ratio

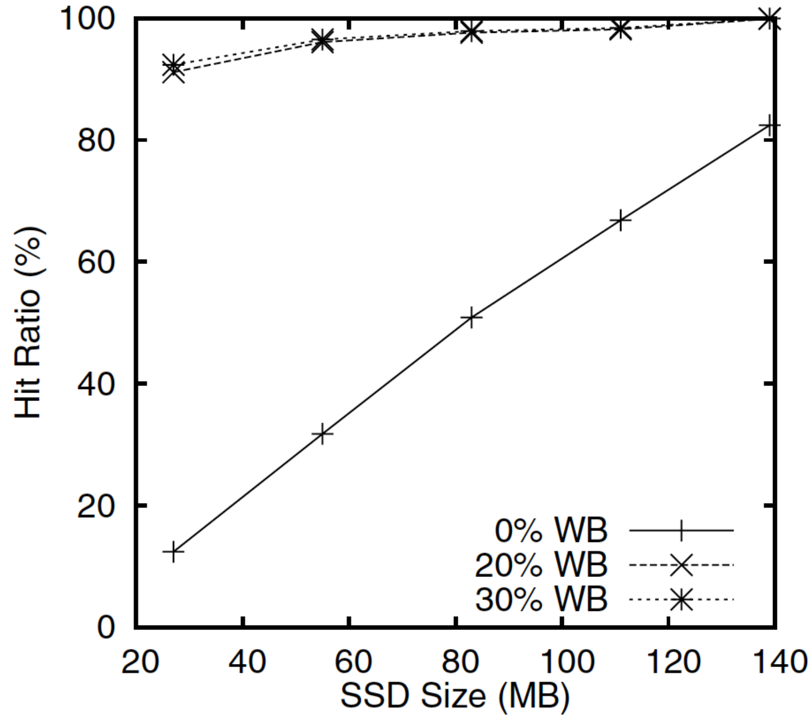
Evaluation

* S. Shah and B. D. Noble. A study of e-mail patterns. In *Software Practice and Experience*, volume 37(14), 2007.

- Benchmark: *Email**
  - intensive synchronous writes with different append sizes and locations based on realistic mail distribution function
  - a more skewed distribution of latencies
  - Most data accesses are small random writes



Execution time

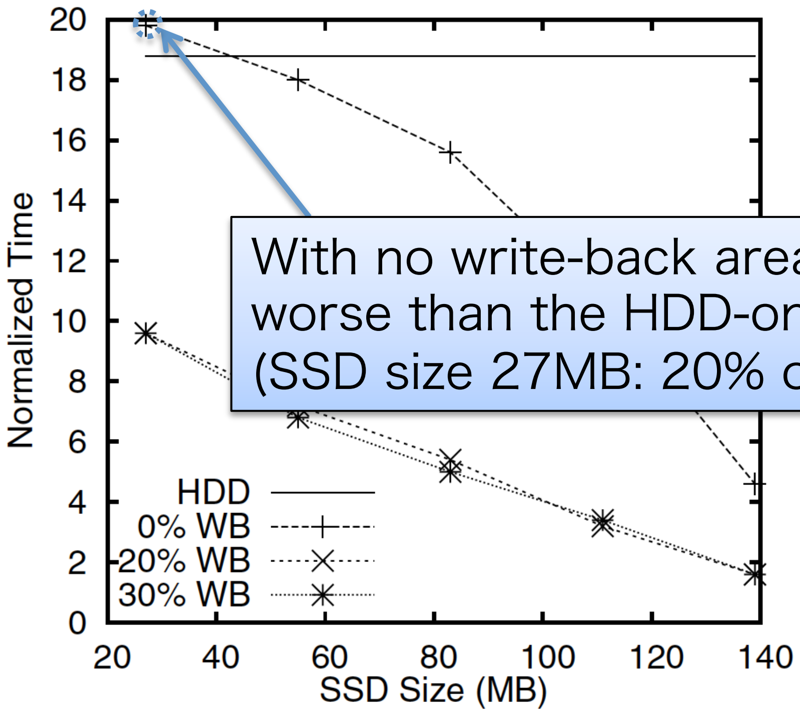


Hit ratio

Evaluation

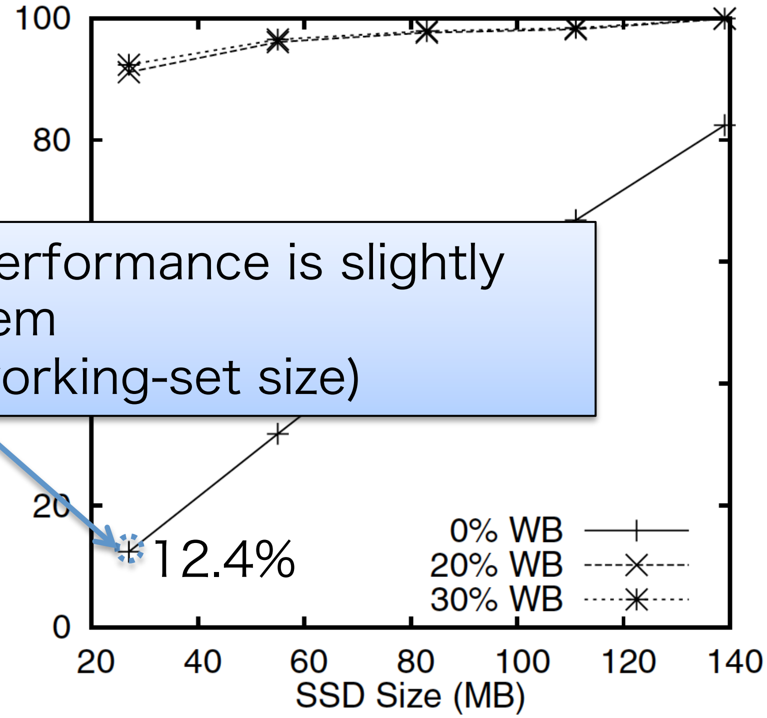
* S. Shah and B. D. Noble. A study of e-mail patterns. In *Software Practice and Experience*, volume 37(14), 2007.

- Benchmark: *Email**
  - intensive synchronous writes with different append sizes and locations based on realistic mail distribution function
  - a more skewed distribution of latencies
  - Most data accesses are small random writes



Execution time

With no write-back area, the performance is slightly worse than the HDD-only system
(SSD size 27MB: 20% of the working-set size)



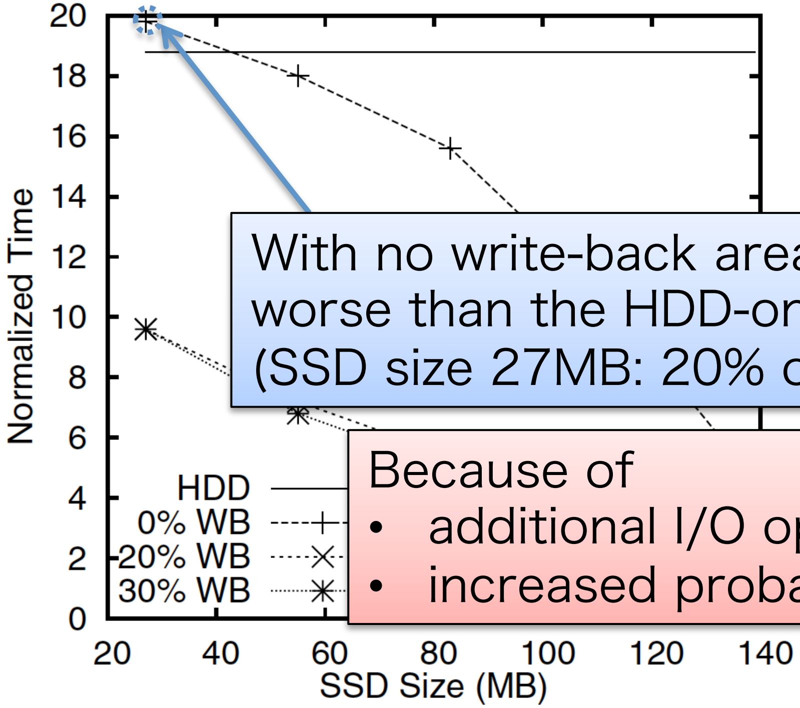
Hit ratio

Evaluation

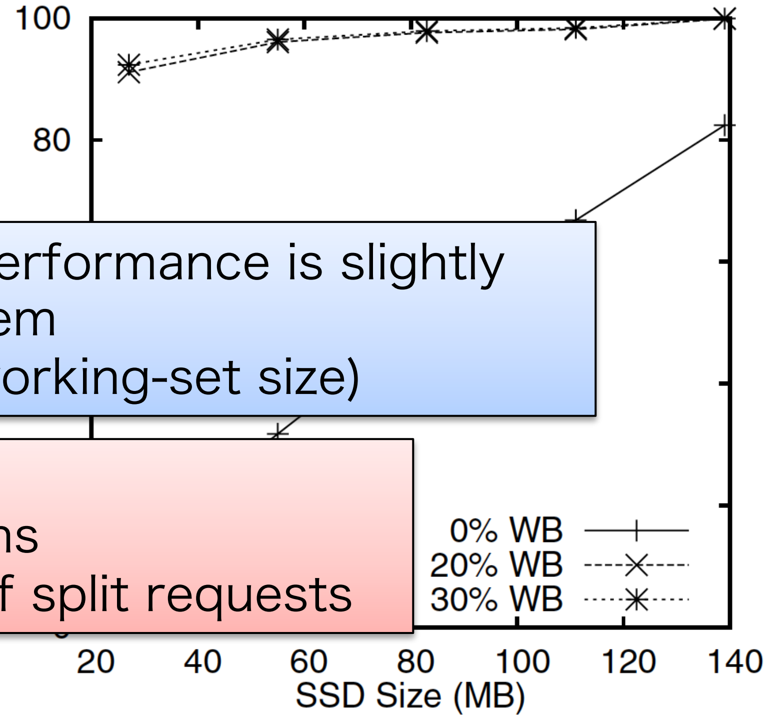
* S. Shah and B. D. Noble. A study of e-mail patterns. In *Software Practice and Experience*, volume 37(14), 2007.

● Benchmark: *Email**

- intensive synchronous writes with different append sizes and locations based on realistic mail distribution function
- a more skewed distribution of latencies
- Most data accesses are small random writes



Execution time



Hit ratio

With no write-back area, the performance is slightly worse than the HDD-only system
(SSD size 27MB: 20% of the working-set size)

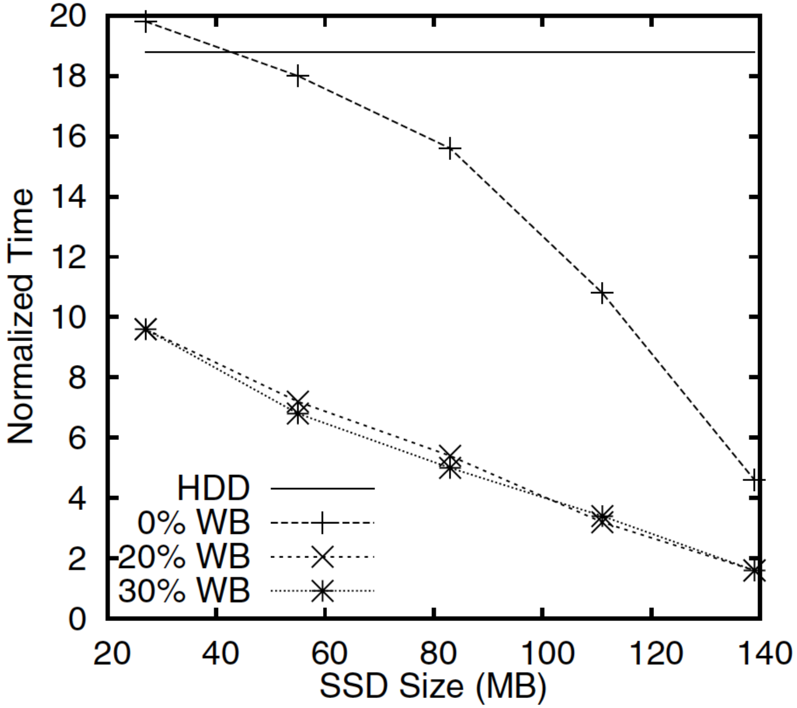
Because of
- additional I/O operations
- increased probability of split requests

Evaluation

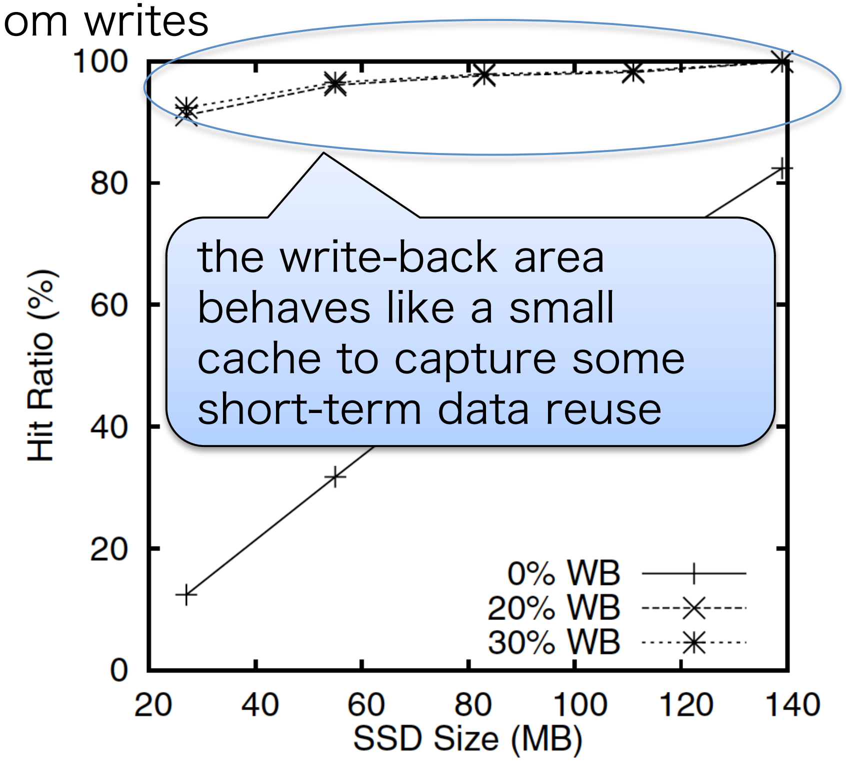
* S. Shah and B. D. Noble. A study of e-mail patterns. In *Software Practice and Experience*, volume 37(14), 2007.

● Benchmark: *Email**

- intensive synchronous writes with different append sizes and locations based on realistic mail distribution function
- a more skewed distribution of latencies
- Most data accesses are small random writes



Execution time

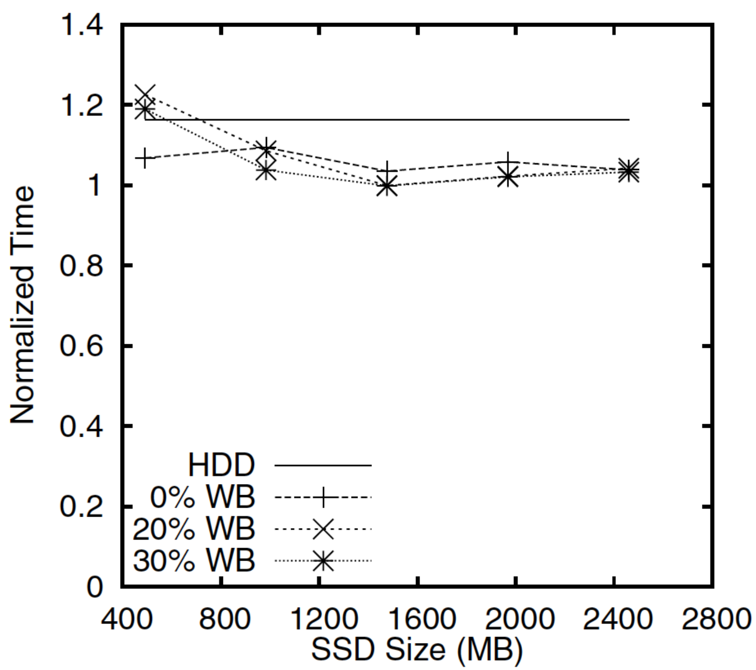


Hit ratio

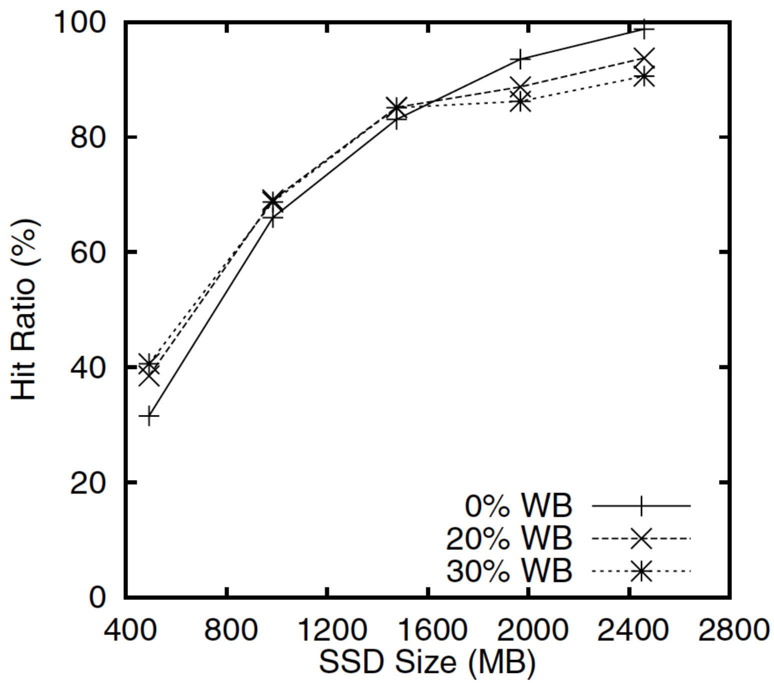
Evaluation

* Transaction Processing Performance Council. TPC Benchmark (2008) http://www.tpc.org/tpch/

- Benchmark: *TPC-H Q1* (query 1 from the TPC-H database benchmark suite)*
  - more sequential data accesses and less I/O intensive than the other workloads



Execution time

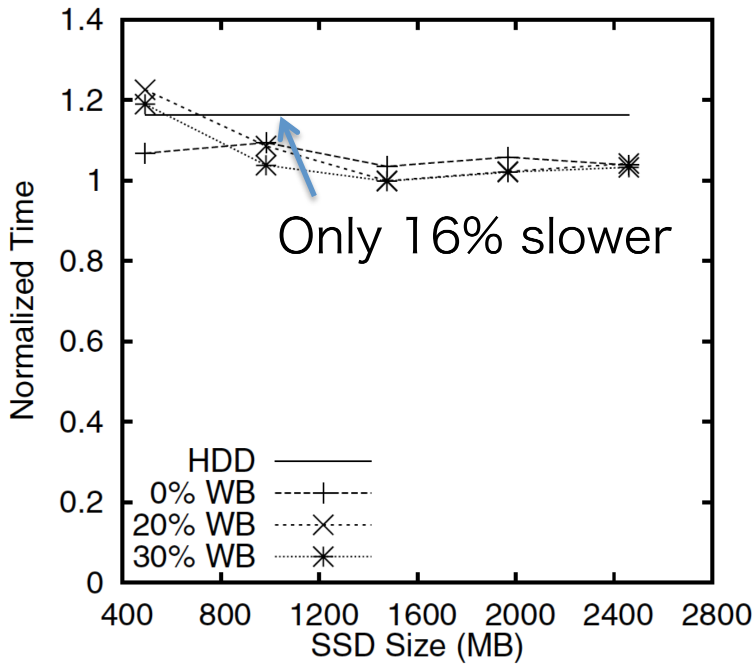


Hit ratio

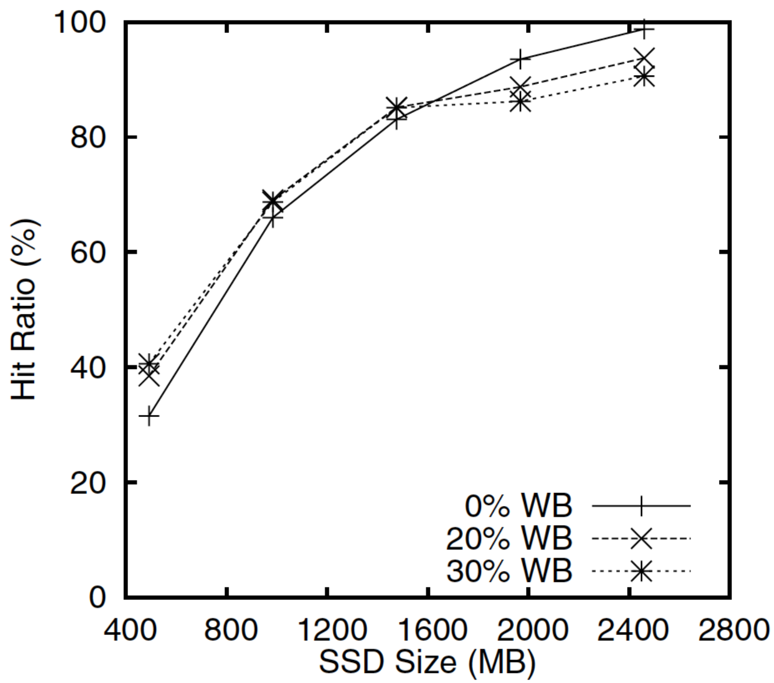
Evaluation

* Transaction Processing Performance Council. TPC Benchmark (2008) http://www.tpc.org/tpch/

- Benchmark: *TPC-H Q1* (query 1 from the TPC-H database benchmark suite)*
  - more sequential data accesses and less I/O intensive than the other workloads



Execution time



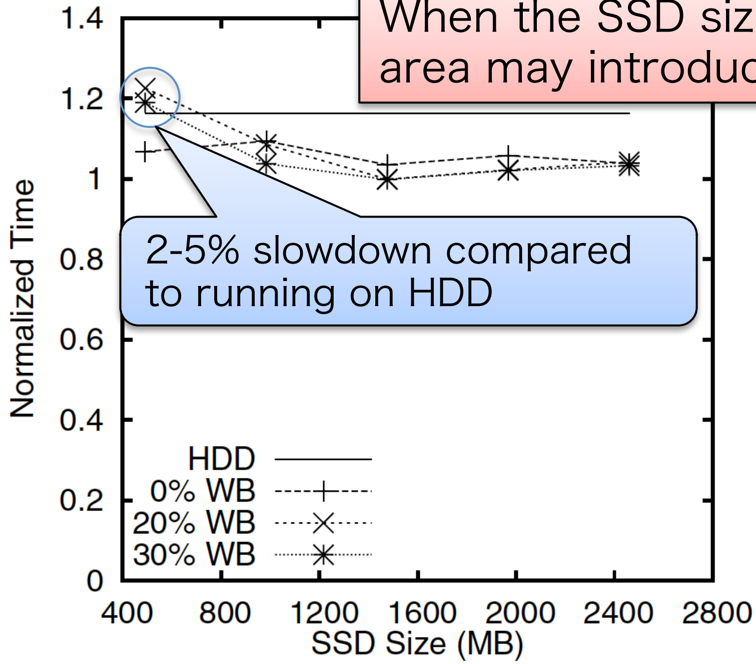
Hit ratio

Evaluation

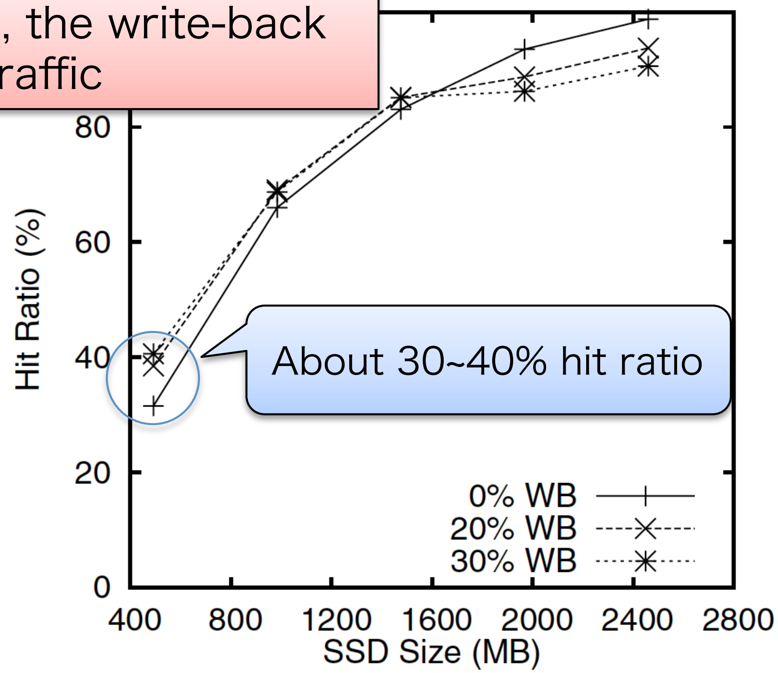
* Transaction Processing Performance Council. TPC Benchmark (2008) http://www.tpc.org/tpch/

- Benchmark: *TPC-H Q1* (query 1 from the TPC-H database benchmark suite)*
  - more sequential data accesses and less I/O intensive than the other workloads

When the SSD size is small, the write-back area may introduce extra traffic



Execution time



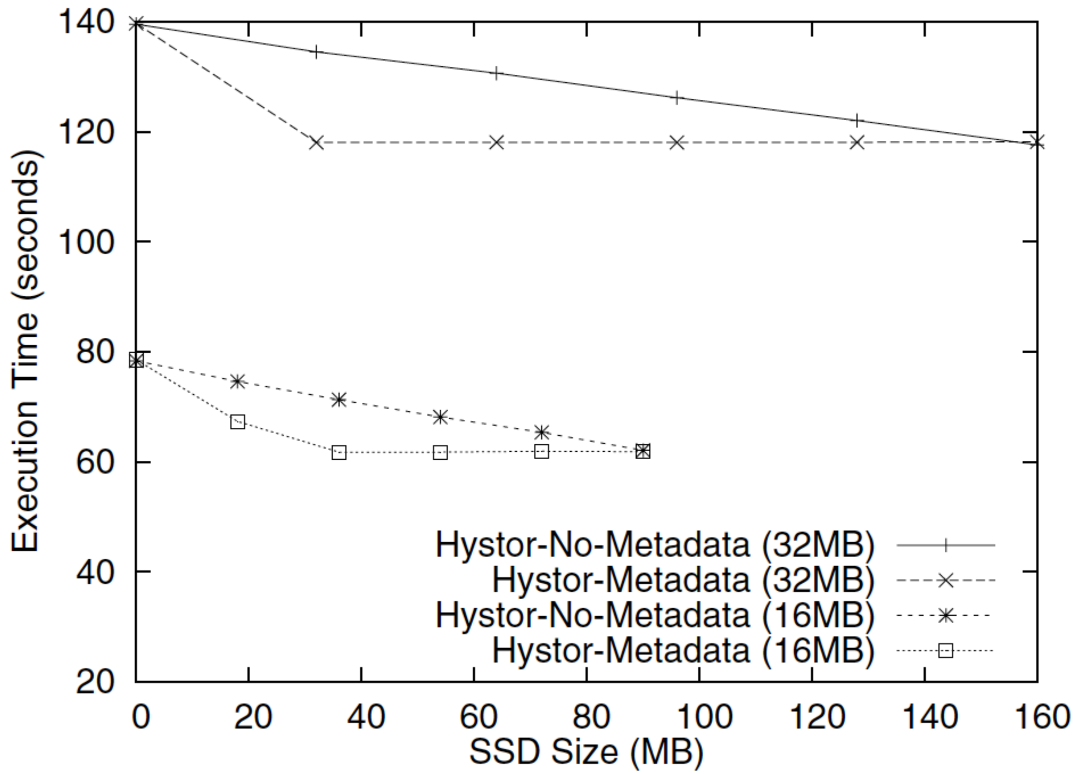
Hit ratio

Evaluation

- Hystor identifies metadata blocks of file systems and remaps them to the SSD
  - How does such an optimization improve performance?
- Comparison the performance of Hystor with and without optimization for file system metadata blocks
  - With optimization: *Hystor-Metadata*
  - Without optimization: *Hystor-No-Metadata*

Evaluation

- Intel® Open Storage Toolkit
  - generating two workloads, which randomly read 4KB data each time until 16MB and 32MB of data are read



- Both approaches eventually can speed up the two workloads by about 20 seconds
- Hystor-Metadata can achieve high performance with a much smaller SSD space
- For the workload reading 32MB data, Hystor-Metadata identifies and remaps nearly all indirect blocks to the SSD with just 32MB of SSD space

Evaluation

- This result shows
  - optimization for metadata blocks can effectively improve system performance with only a small amount of SSD space
    - especially for metadata-intensive workloads

  - high-cost cold misses can be avoided
    - due to proactively identifying these *semantically critical* blocks (file system metadata blocks) at an early stage

Evaluation

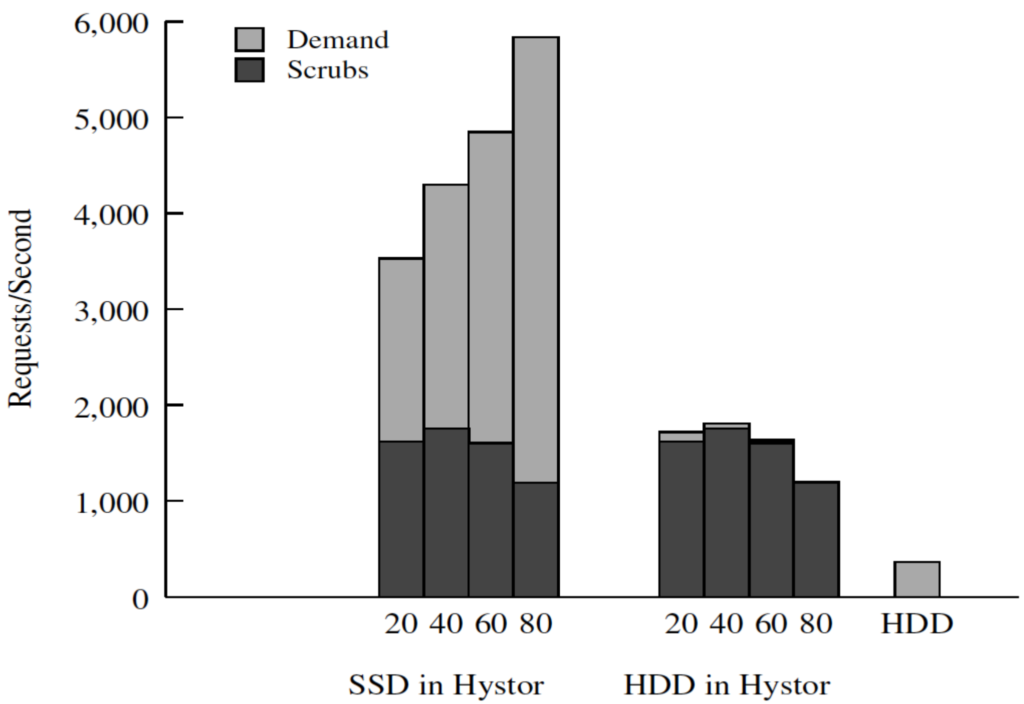
- *Scrubbing* - Dirty blocks buffered in the write-back area have to be written back to the HDD in the background

- Each scrub operation can cause two additional I/O operations
  - A read from the SSD
  - A write to the HDD

- How does scrubbing affect performance?
  - Here, *email* is used for the evaluation
    - Because of the worst case for scrubs

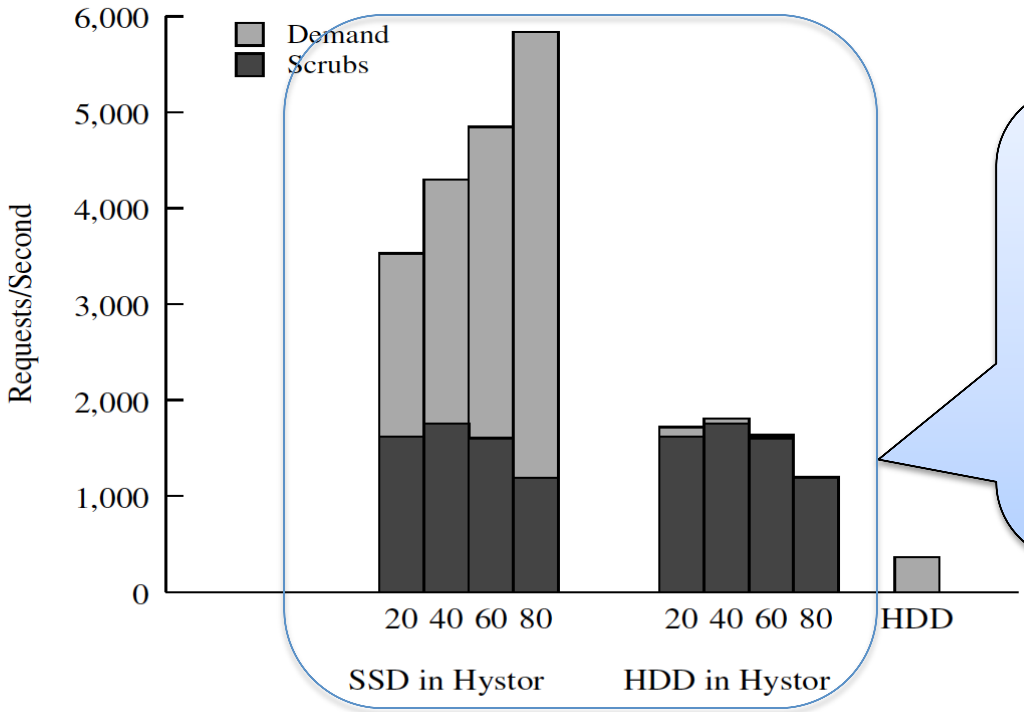
Evaluation

- X-axis: Various configurations of the SSD size (% of the working-set size) and HDD-only system
- Y-axis: Request arrival rate in *email*
  - Demand: requests by upper-layer components
  - Scrubs: requests by internal scrubbing daemon



Evaluation

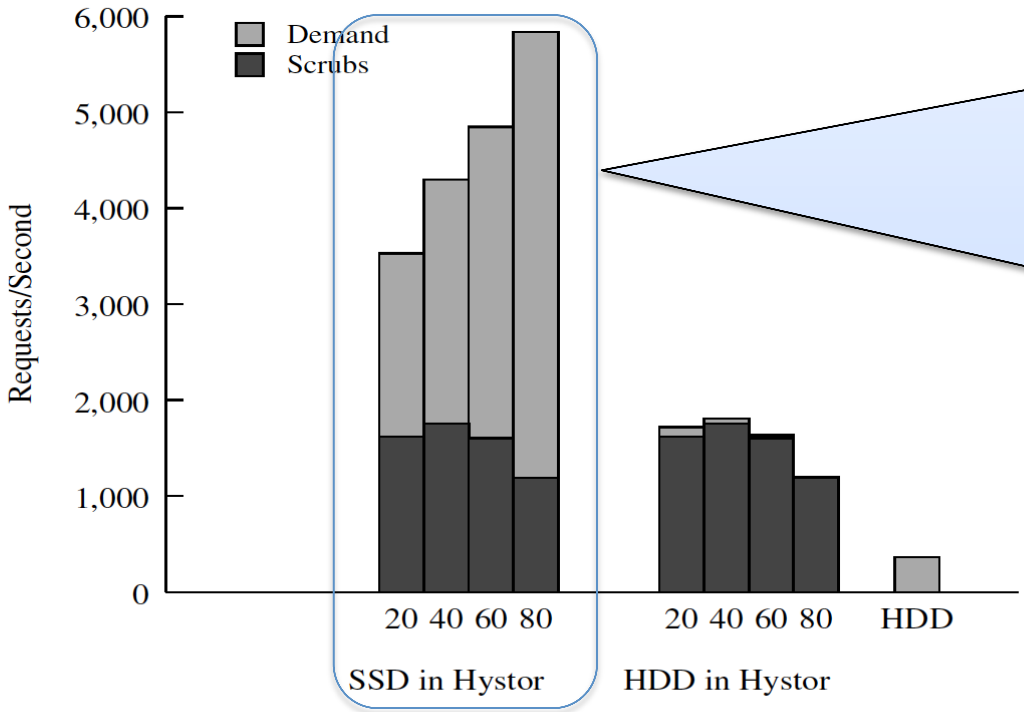
- X-axis: **Various configurations of the SSD size (% of the working-set size) and HDD-only system**
- Y-axis: Request arrival rate in *email*
  - Demand: requests by upper-layer components
  - Scrubs: requests by internal scrubbing daemon



High rate reasons
- a large request in Hystor may split into several small ones to different devices
- two additional I/O operations are needed for each scrub

Evaluation

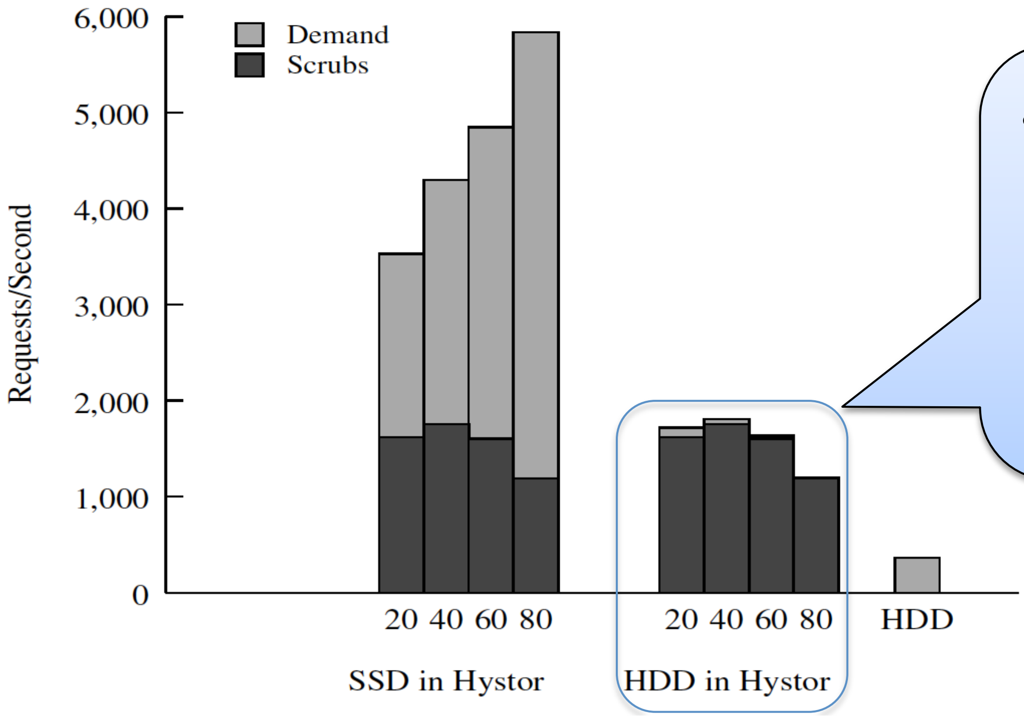
- X-axis: **Various configurations of the SSD size (% of the working-set size) and HDD-only system**
- Y-axis: Request arrival rate in *email*
  - Demand: requests by upper-layer components
  - Scrubs: requests by internal scrubbing daemon



- Increasing to 80% of the working-set size, the arrival rate of scrub requests drops by nearly 25% on the SSD due to less frequent scrubbing
- The arrival rate of demand requests increases
  - Reduction of execution time
  - The # of demand requests remains unchanged

Evaluation

- X-axis: **Various configurations of the SSD size (% of the working-set size) and HDD-only system**
- Y-axis: Request arrival rate in *email*
  - Demand: requests by upper-layer components
  - Scrubs: requests by internal scrubbing daemon



- These requests happen in the background
  - The performance impact on the foreground jobs is minimal

Evaluation

- This result shows
  - Although a considerable increase of request arrival rate is resident on both storage devices, conducting background scrubbing causes minimal performance impact, even for write-intensive workloads.

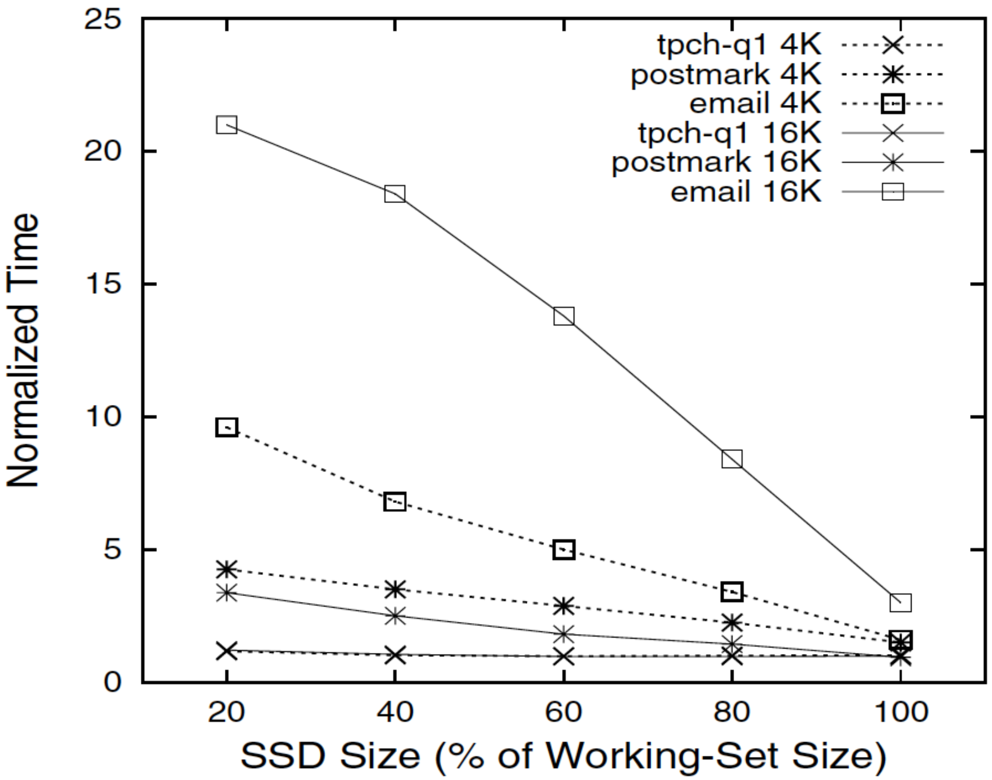
Evaluation

- Chunk size
  - Large: desirable for reducing memory overhead of the mapping table and the block table
  - Small: effectively improving utilization of the SSD space
    - a large chunk may contain both hot and cold data

- So, how does chunk size affect performance?

Evaluation

- Chunk size: 4KB(8 sector), 16KB(32 sector)
- Write-back fraction: 20%



- With a large chunk size (16KB), the performance of *email* degrades significantly
  - most of the requests in *email* are small
  - hot and cold data could co-exist in a large chunk → miss rate increases

Evaluation

- This result shows
  - For a small-capacity SSD
    - a small chunk size should be used to avoid wasting precious SSD space
  - For a large-capacity SSD
    - It's possible to use a large chunk size and afford the luxury of increased internal fragmentation in order to reduce overhead
- In general
  - a small chunk size (e.g. 4KB) is normally sufficient for optimizing performance
    - So is Hystor (default 4KB)

Agenda

- Introduction
- SSD Performance Advantages
- High-Cost Data Blocks
- Maintaining Data Access History
- The Design and Implementation of Hystor
- Evaluation
- Conclusion and Impression ☜

Conclusion

- Need to find the fittest position of SSDs in the existing systems to strike a right balance between performance and cost
- This work shows
  - It's possible to identify the data that are best suitable to be held in SSD by using a simple yet effective metric
  - High-cost data blocks can be efficiently maintained in the block table at a low cost
  - SSDs should play a major role in the storage hierarchy by adaptively and timely retaining performance- and semantically-critical data
  - It's also effective to use SSD as a write-back buffer for incoming write requests
  - Hystor can effectively leverage the performance merits of SSDs with minimized system changes

Impression

- Pros
  - Exploratory evaluations were executed in detail
    - E.g. SSD performance, Indicator Metric…
  - A lot of detailed evaluation results about Hystor
  - Simple yet smart approach to improve system performance
- Cons
  - Few figures (Section5, Section6)
  - I would like to know how different a hardware implementation is