The Impact of Solid State Drive on Search Engine Cache Management

Jiancong Tong
Ph.D. candidate at Nankai University
Visiting student at University of Melbourne
lingfenghx@gmail.com

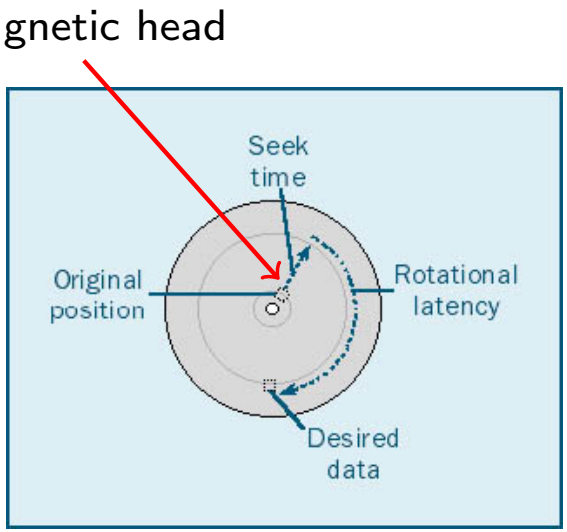
with
Gang Wang, Xiaoguang Liu (Nankai University)
and
Jianguo Wang, Eric Lo, Man Lung Yiu (Hong Kong Polytechnic University)

Monash University
May 12, 2014

Outline

Hard Disk Drive

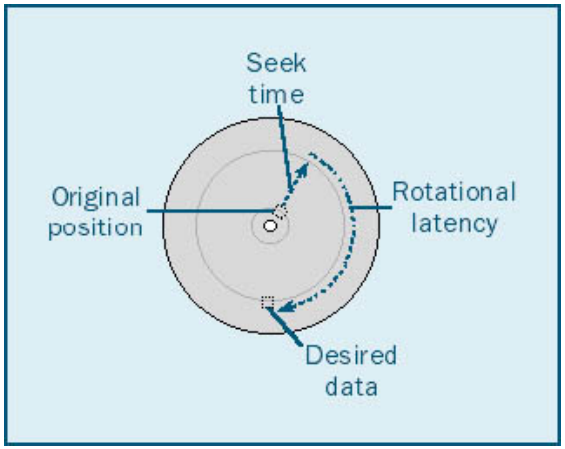
Hard Disk Drive (HDD)



How does HDD work? [Garcia et al., 2000]

Hard Disk Drive

Hard Disk Drive (HDD)



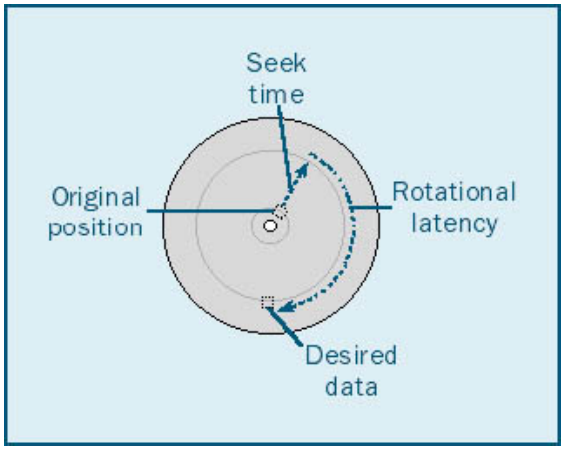
How does HDD work? [Garcia et al., 2000]

Random read latency of HDD

- Seek time
- Rotational latency
- Transfer time

Hard Disk Drive

Hard Disk Drive (HDD)



How does HDD work? [Garcia et al., 2000]

Random read latency of HDD

- Seek time
- Rotational latency
- Transfer time

Caching technology is used to reduce the latency.

What is a Cache?

Small, fast memory used to improve average access time to large, slow storage media.

What is a Cache?

Small, fast memory used to improve average access time to large, slow storage media.

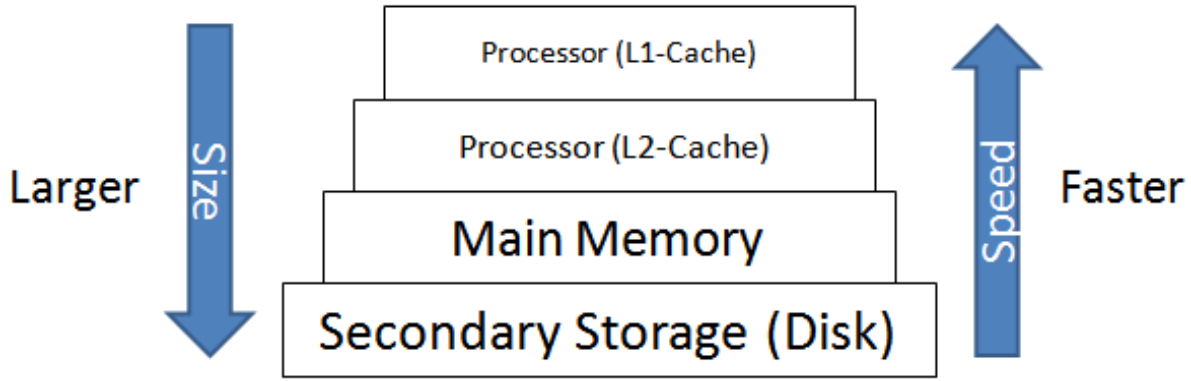
Exploits locality: both spacial and temporal.

What is a Cache?

Small, fast memory used to improve average access time to large, slow storage media.

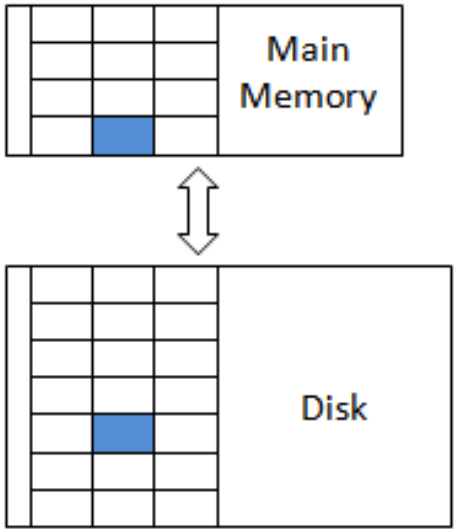
Exploits locality: both spacial and temporal.

Almost everything is a cache in computer architecture...

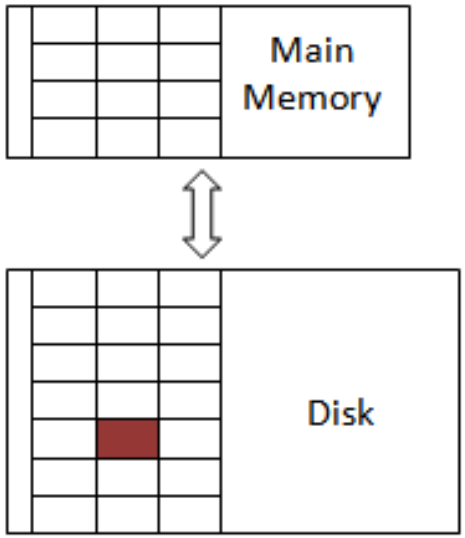


What is a Cache? (Cont.)

- *Cache Hit*: the requested data is found in the memory
- *Cache Miss*: the requested data is not found in the memory



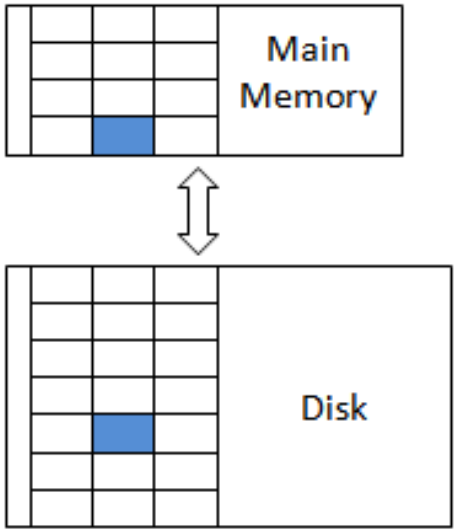
Cache hit



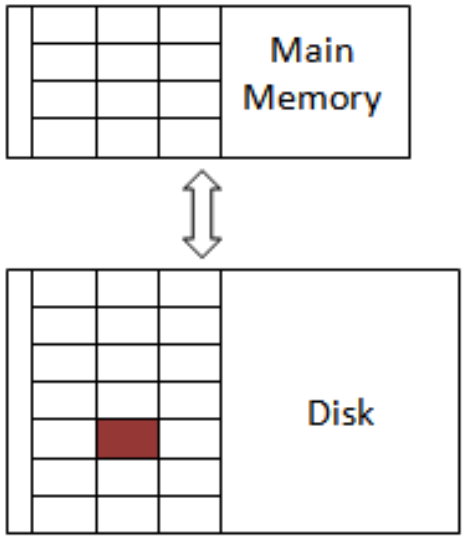
Cache miss

What is a Cache? (Cont.)

- *Cache Hit*: the requested data is found in the memory
- *Cache Miss*: the requested data is not found in the memory



Cache hit

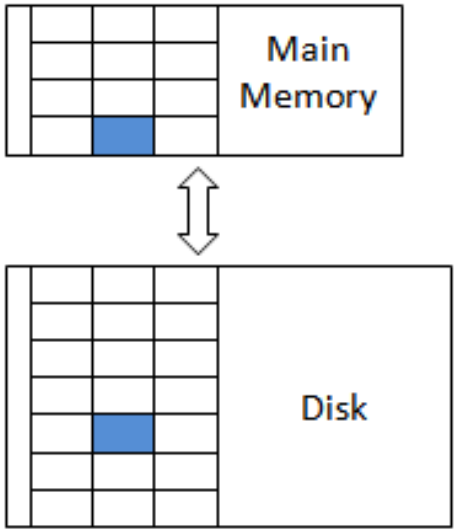


Cache miss

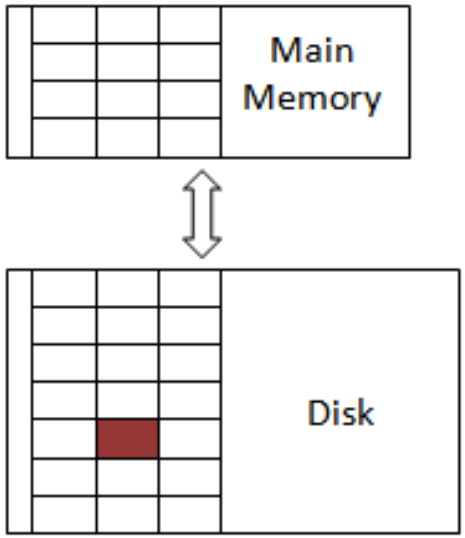
$$\text{Hit ratio} = \frac{\#\text{Hits}}{\#\text{Memory accesses}}$$

What is a Cache? (Cont.)

- *Cache Hit*: the requested data is found in the memory
- *Cache Miss*: the requested data is not found in the memory



Cache hit

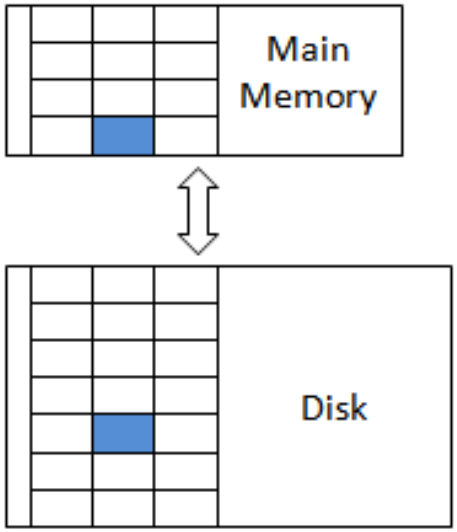


Cache miss

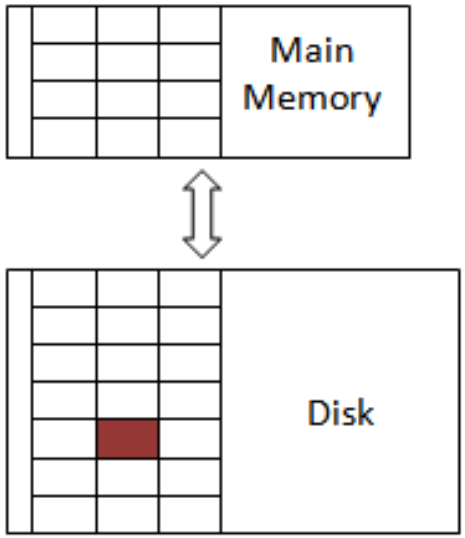
$$\text{Hit ratio} = \frac{\#\text{Hits}}{\#\text{Memory accesses}} \qquad \text{Miss ratio} = \frac{\#\text{Misses}}{\#\text{Memory accesses}}$$

What is a Cache? (Cont.)

- *Cache Hit*: the requested data is found in the memory
- *Cache Miss*: the requested data is not found in the memory



Cache hit



Cache miss

$$\text{Hit ratio} = \frac{\#\text{Hits}}{\#\text{Memory accesses}} \qquad \text{Miss ratio} = \frac{\#\text{Misses}}{\#\text{Memory accesses}}$$

$$\text{Hit ratio} + \text{Miss ratio} = 1$$

What is Solid State Drive?

Hard Disk Drive (HDD)
with magnetic moving head

Solid State Drive (SSD)
based on semiconductor chips

- SSD: **New Faster ($10 \sim 100$x)** HDD with compatible interface

What is Solid State Drive?

Hard Disk Drive (HDD)
with magnetic moving head

Solid State Drive (SSD)
based on semiconductor chips

- SSD: **New Faster ($10 \sim 100$x)** HDD with compatible interface
- Strong technical merits: [Chen et al., 2009]
  1. Lower power consumption
  2. More compact size
  3. Better shock resistance
  4. Extraordinarily faster random data access

The Rise of SSD

Words of Pioneer (by Jim Gray, 2006)

Tape is dead; Disk is tape; Flash is disk; RAM locality is King.

The Rise of SSD

Words of Pioneer (by Jim Gray, 2006)

Tape is dead; Disk is tape; Flash is disk; RAM locality is King.

SSD in Large-Scale System Architectures

- **Google** 2008 (or later)
- **Baidu** 2008
- Facebook 2010
- Myspace 2010
- Oracle 2011
- Microsoft Azure 2012

The Rise of SSD

Words of Pioneer (by Jim Gray, 2006)

Tape is dead; Disk is tape; Flash is disk; RAM locality is King.

SSD in Large-Scale System Architectures

- **Google** 2008 (or later)
- **Baidu** 2008
- Facebook 2010
- Myspace 2010
- Oracle 2011
- Microsoft Azure 2012

Trends

Flash memory based SSD is replacing and is going to completely replace HDD as the major storage medium!

Challenges

Existing caching policies were originally designed for HDD

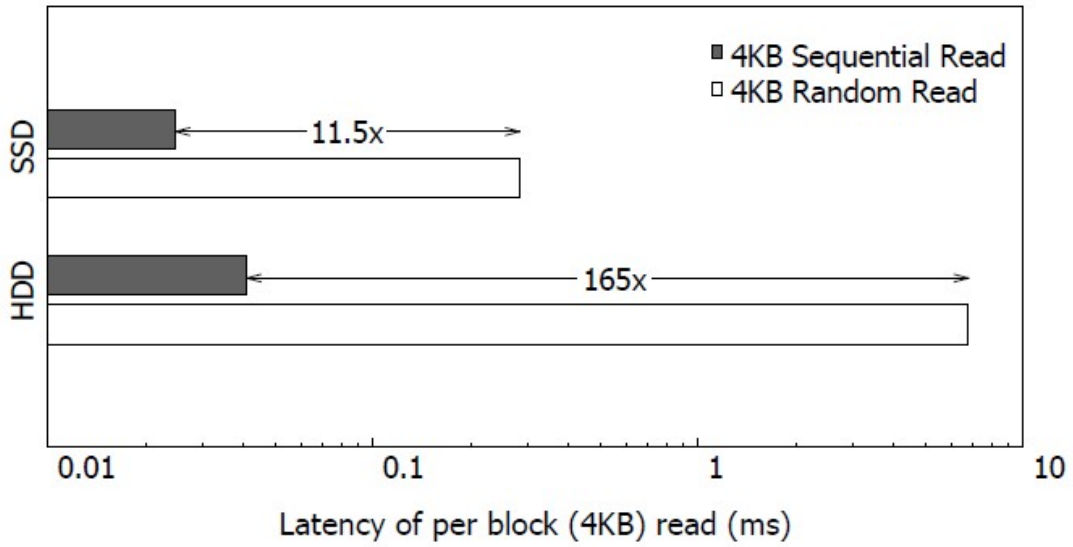
- HDD: Very slow random read (compare to sequential read)
- Cache design principle: minimize random read

Challenges

Existing caching policies were originally designed for HDD

- HDD: Very slow random read (compare to sequential read)
- Cache design principle: minimize random read

However...



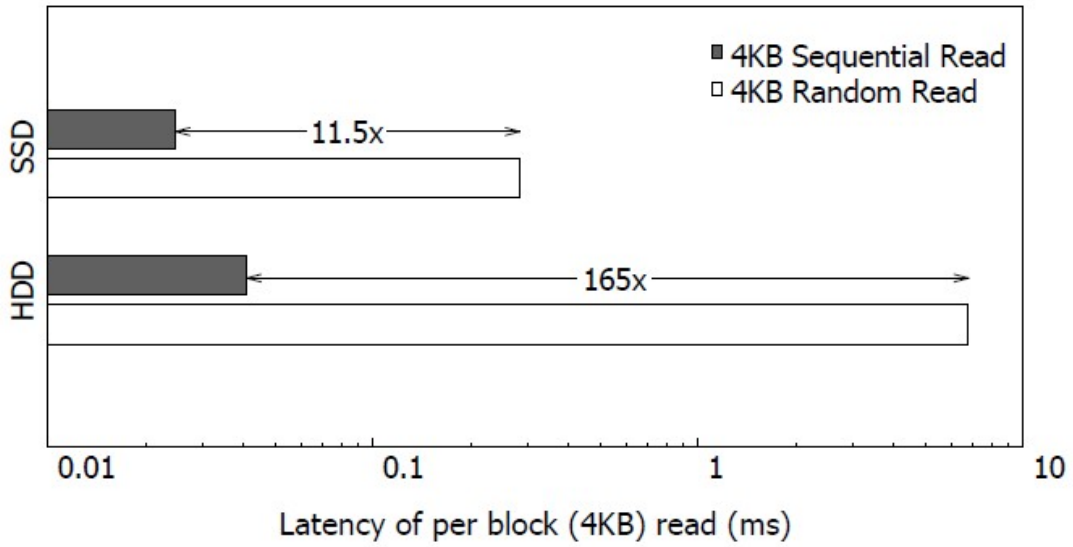
[Tong et al., 2013]

Challenges

Existing caching policies were originally designed for HDD

- HDD: Very slow random read (compare to sequential read)
- Cache design principle: minimize random read

However...



[Tong et al., 2013]

What now?

Research Questions

RQ1: What is the impact of SSD on buffer management?

Research Questions

RQ1: What is the impact of SSD on buffer management?

- Are the existing cache techniques designed for HDD-based search engine still good for SSD-based search engine?

Research Questions

RQ1: What is the impact of SSD on buffer management?

- Are the existing cache techniques designed for HDD-based search engine still good for SSD-based search engine?
- What measure(s) should be used to define a 'good' cache policy in this case?

Research Questions

RQ1: What is the impact of SSD on buffer management?

- Are the existing cache techniques designed for HDD-based search engine still good for SSD-based search engine?
- What measure(s) should be used to define a 'good' cache policy in this case?
- If the performance of caching is improved or degraded, what does that mean to the entire system?

Research Questions

RQ1: What is the impact of SSD on buffer management?

- Are the existing cache techniques designed for HDD-based search engine still good for SSD-based search engine?
- What measure(s) should be used to define a 'good' cache policy in this case?
- If the performance of caching is improved or degraded, what does that mean to the entire system?

RQ2: How could we deal with that?

Research Questions

RQ1: What is the impact of SSD on buffer management?

- Are the existing cache techniques designed for HDD-based search engine still good for SSD-based search engine?
- What measure(s) should be used to define a 'good' cache policy in this case?
- If the performance of caching is improved or degraded, what does that mean to the entire system?

RQ2: How could we deal with that?

- What to do if the efficiency of the entire system is affected by such impact?

Research Questions

RQ1: What is the impact of SSD on buffer management?

- Are the existing cache techniques designed for HDD-based search engine still good for SSD-based search engine?
- What measure(s) should be used to define a 'good' cache policy in this case?
- If the performance of caching is improved or degraded, what does that mean to the entire system?

RQ2: How could we deal with that?

- What to do if the efficiency of the entire system is affected by such impact?
- Can we propose better cache policies for SSD-based systems?

1 Background and Motivation

2 Research Questions and Answers

- RQ1: What is the impact of SSD on buffer management?
- RQ2: How could we deal with that?

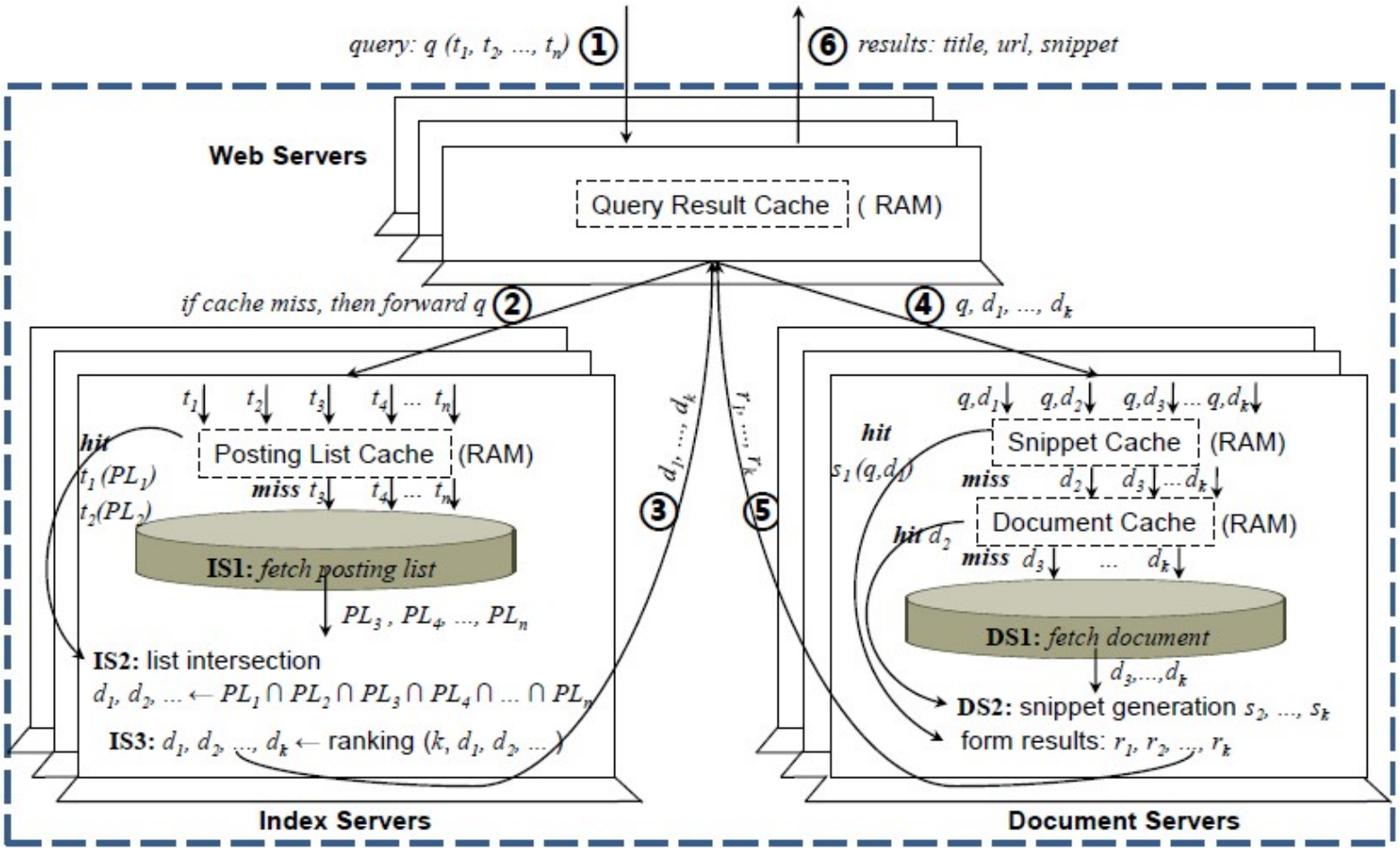
Large-scale experimental study

RQ1 can be answered by evaluating the effectiveness of existing caching policies on an SSD-based search engine.

Settings

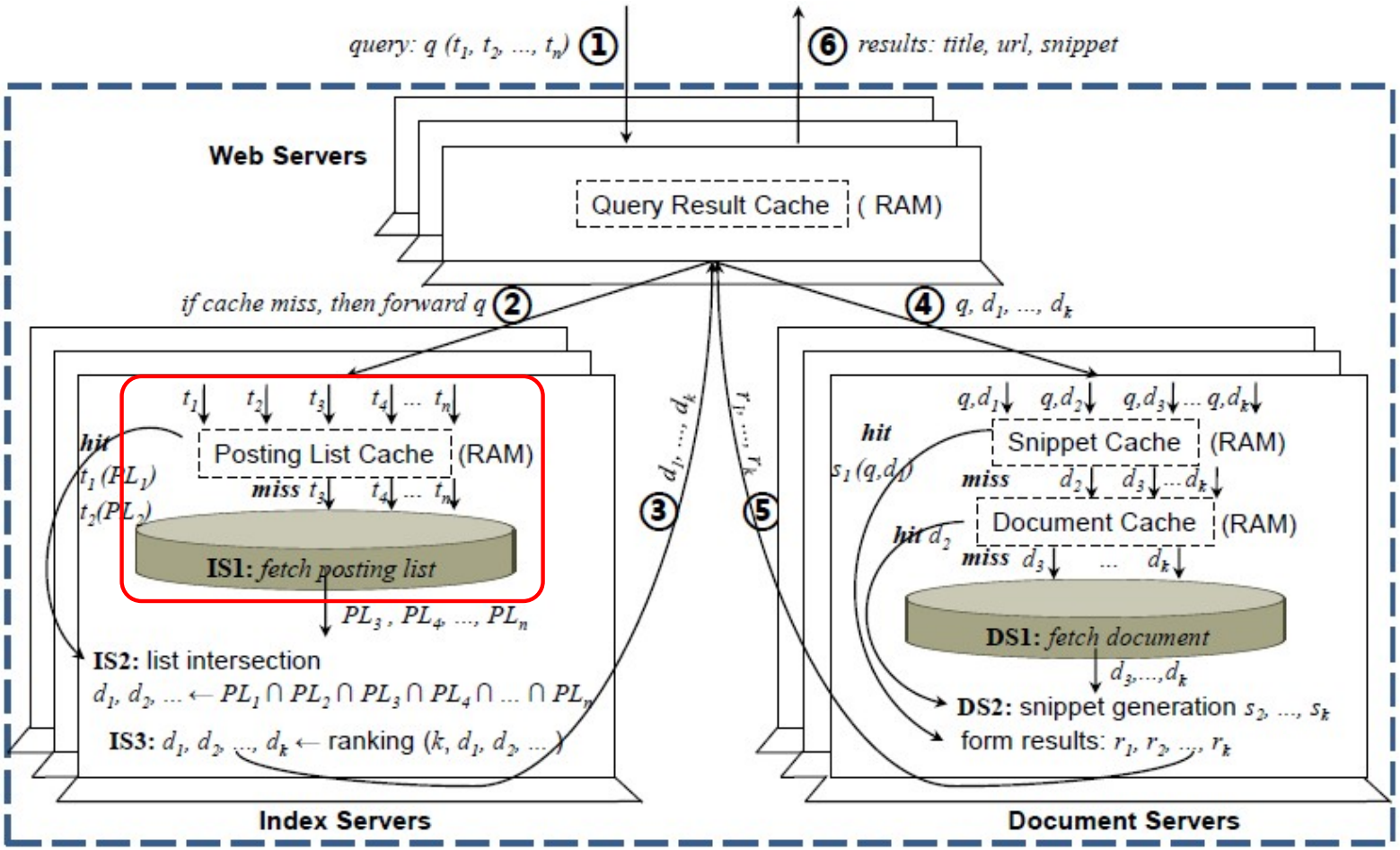
- Datasets
  - Web documents: 12,000,000 (∼100GB)
  - Queries: 1,000,000
- Device
  - SSD: ADATA 256GB SSD
  - HDD: Seagate 3TB 7200rpm
- System: Apache Lucene
- Measure: Query time (NOT hit ratio)

Overview of search engine architecture



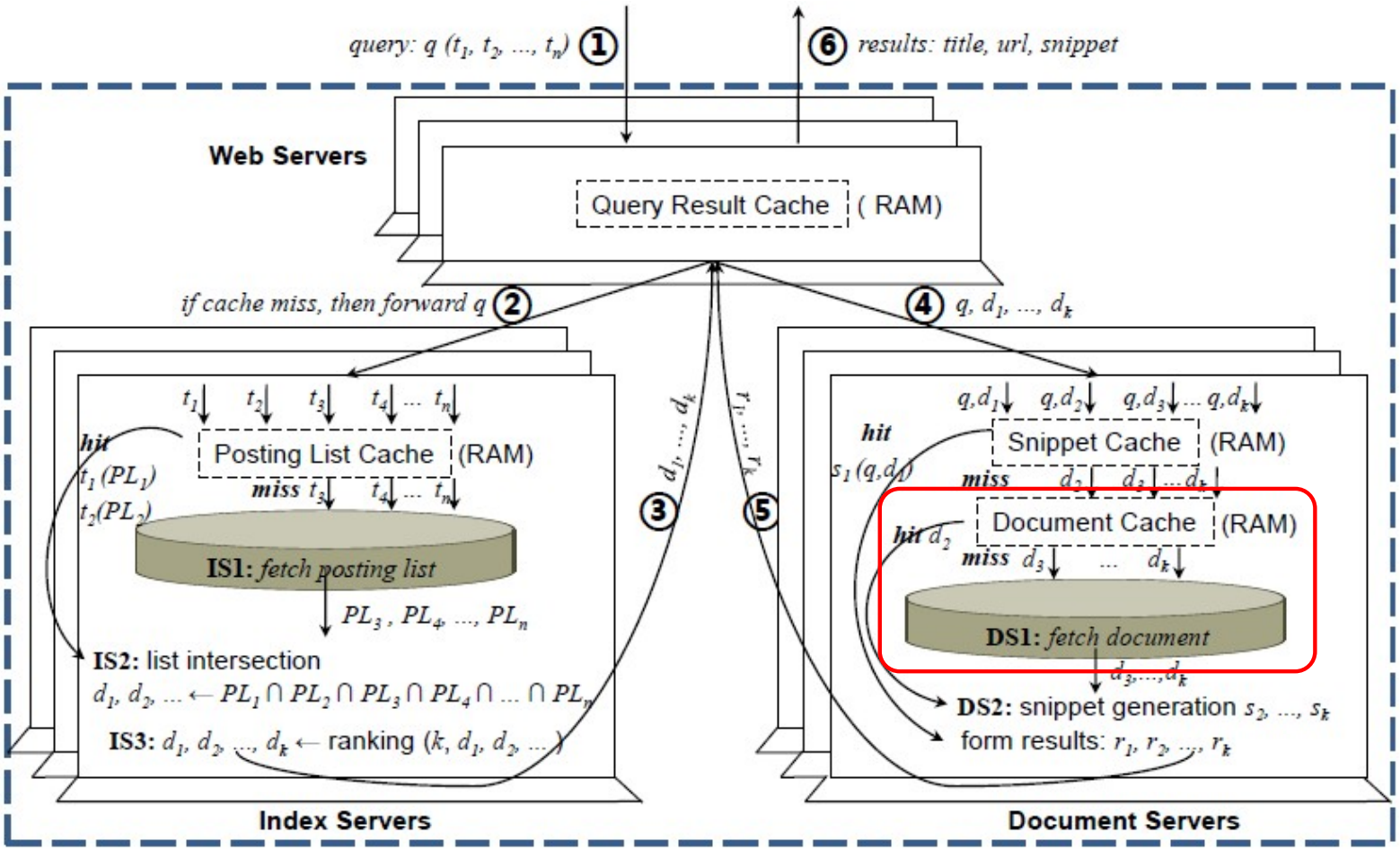
[Wang et al., 2013]

Overview of search engine architecture



[Wang et al., 2013]

Overview of search engine architecture



[Wang et al., 2013]

Index Server and List (Index) Cache

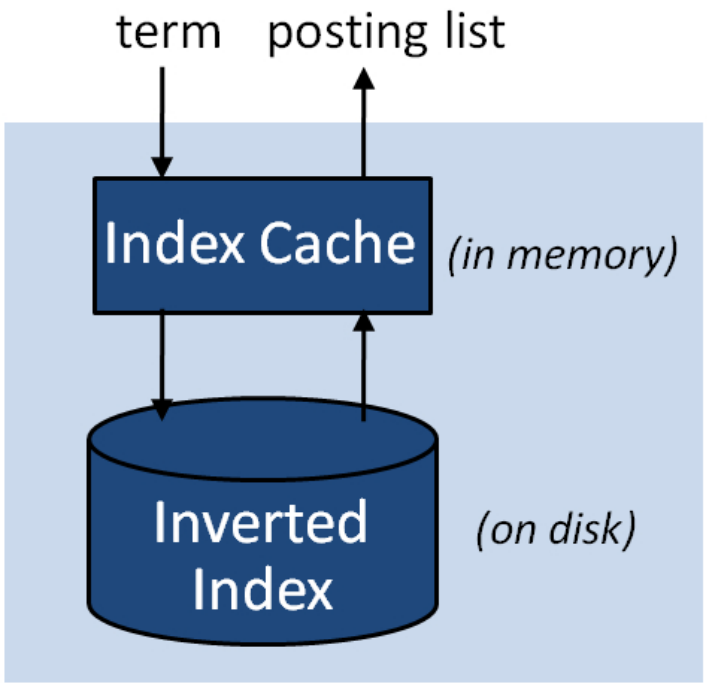
Posting Lists (Inverted Lists)

- `Clayton` (2, 4, 5, 10, 100, 107, ..., 1,000,000)
- `Monash` (1, 2, 3, 23, ...)

Index Server and List (Index) Cache

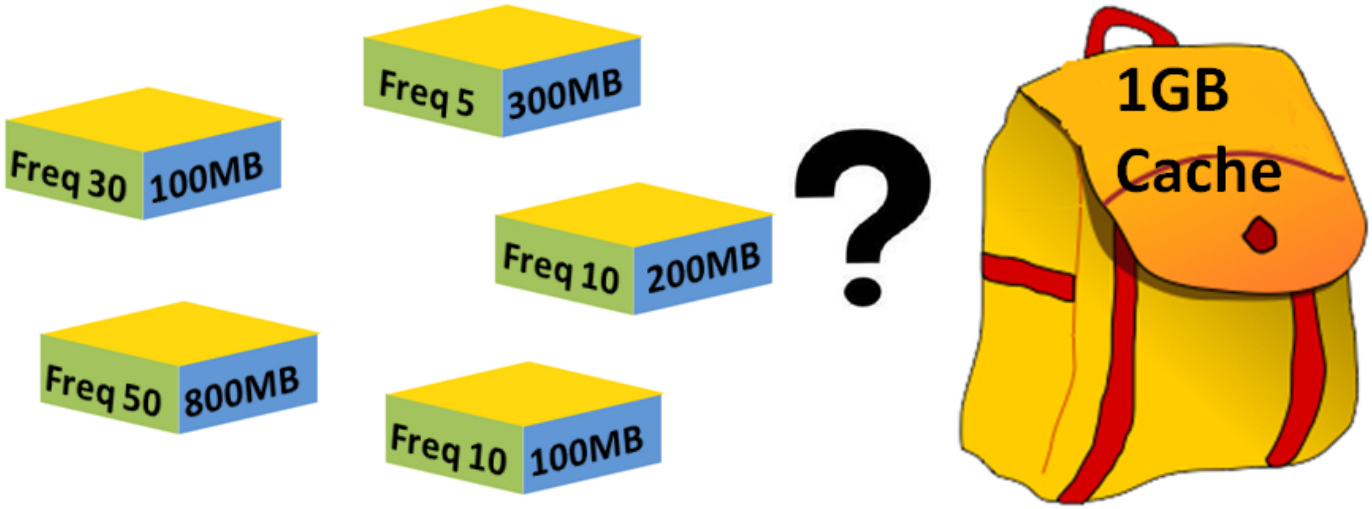
Posting Lists (Inverted Lists)

- `Clayton` (2, 4, 5, 10, 100, 107, ..., 1,000,000)
- `Monash` (1, 2, 3, 23, ...)

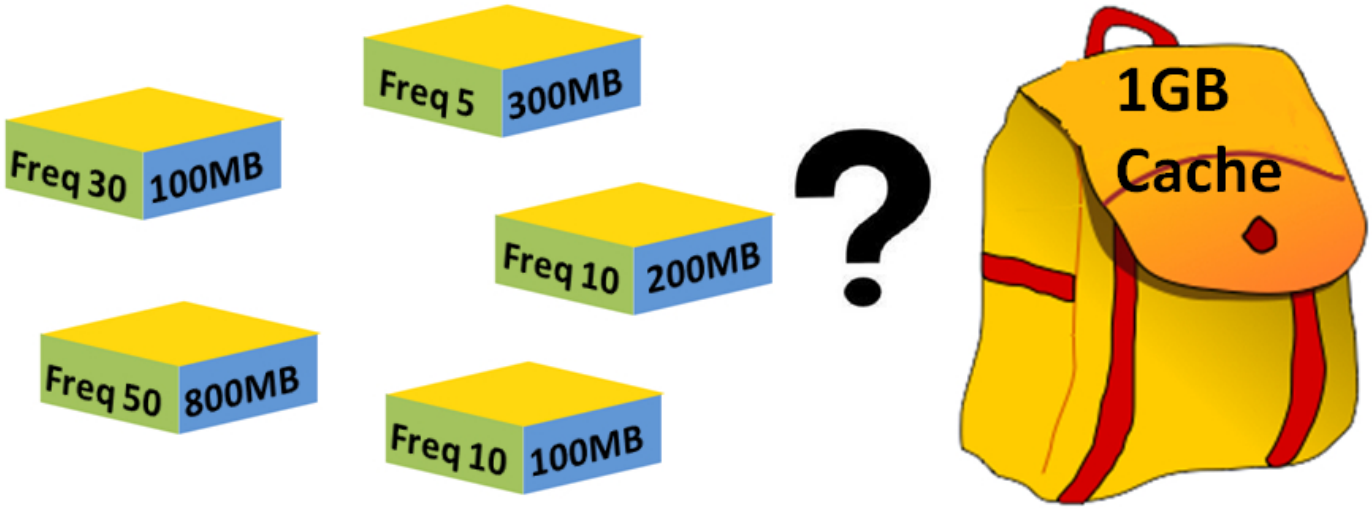


Index Server

Which lists should be kept in the cache?



Which lists should be kept in the cache?



Metrics to consider

- High Frequency First?
- High Frequency/Size First?

Previous conclusion [Baeza-Yates et al., 2007]

Frequency/Size is a better metric (for HDD).

Previous conclusion [Baeza-Yates et al., 2007]

Frequency/Size is a better metric (for HDD).

The reason

On HDD

- Random read is 130 $\sim$ 170x slower than sequential read
  $\Longrightarrow$ List access time is (almost) a constant!

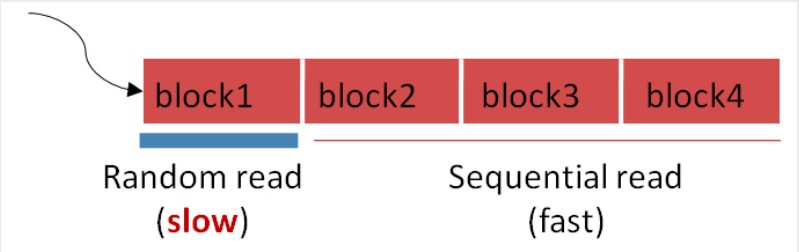
Previous conclusion [Baeza-Yates et al., 2007]

Frequency/Size is a better metric (for HDD).

The reason

On HDD

- Random read is 130 $\sim$ 170x slower than sequential read
  $\Longrightarrow$ List access time is (almost) a constant!



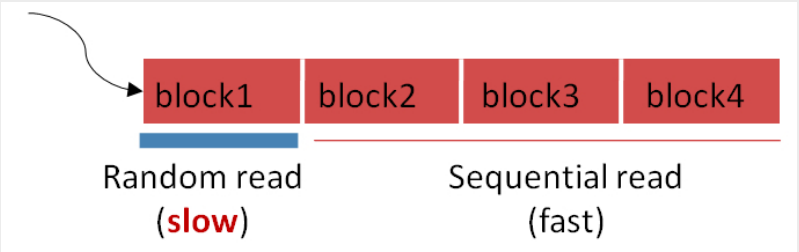
Previous conclusion [Baeza-Yates et al., 2007]

Frequency/Size is a better metric (for HDD).

The reason

On HDD

- Random read is 130 $\sim$ 170x slower than sequential read
  $\Longrightarrow$ List access time is (almost) a constant!



- It is #access that dominates, not |access data|
  $\Longrightarrow$ Short lists are favored (more lists can be held)

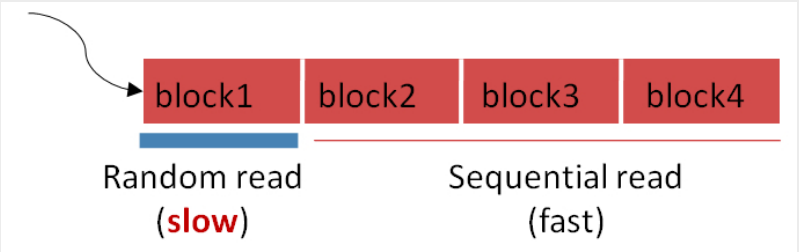
Previous conclusion [Baeza-Yates et al., 2007]

Frequency/Size is a better metric (for HDD).

The reason

On HDD

- Random read is 130 $\sim$ 170x slower than sequential read
  $\Longrightarrow$ List access time is (almost) a constant!



- It is #access that dominates, not |access data|
  $\Longrightarrow$ Short lists are favored (more lists can be held)

- High benefit gained when size is taken into account

How does SSD change the story? [Wang et al., 2013]

Differences

On SSD

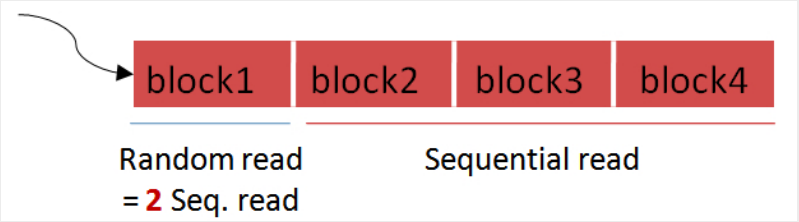
- ~~Random read is 130 $\sim$ 170x slower than sequential read~~
  ~~$\Longrightarrow$ List access time is (almost) a constant!~~
- Random read is only 2 $\sim$ 10x slower than sequential read
  $\Longrightarrow$ List access time varies a lot!

How does SSD change the story? [Wang et al., 2013]

Differences

On SSD

- ~~Random read is 130 $\sim$ 170x slower than sequential read~~
  ~~$\Longrightarrow$ List access time is (almost) a constant!~~
- Random read is only 2 $\sim$ 10x slower than sequential read
  $\Longrightarrow$ List access time varies a lot!

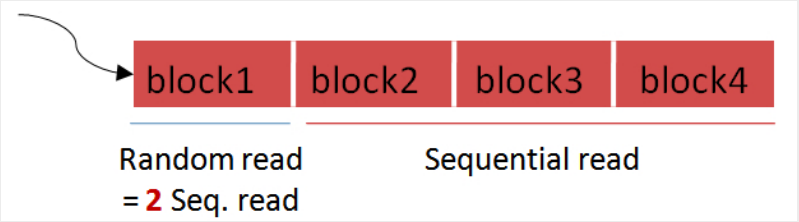


How does SSD change the story? [Wang et al., 2013]

Differences

On SSD

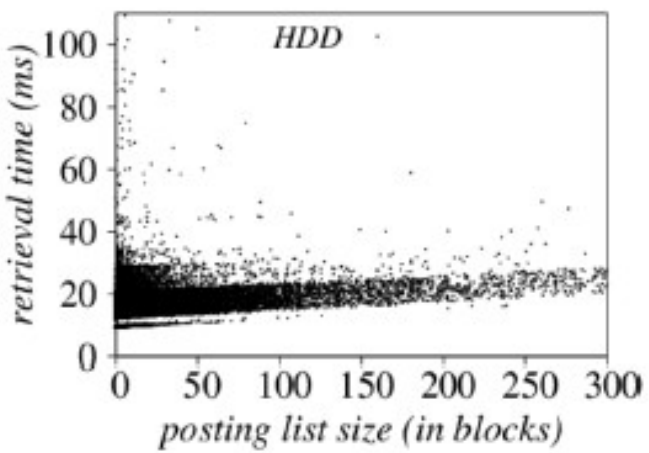
- ~~Random read is 130 $\sim$ 170x slower than sequential read~~
  ~~$\Longrightarrow$ List access time is (almost) a constant!~~
- Random read is only 2 $\sim$ 10x slower than sequential read
  $\Longrightarrow$ List access time varies a lot!



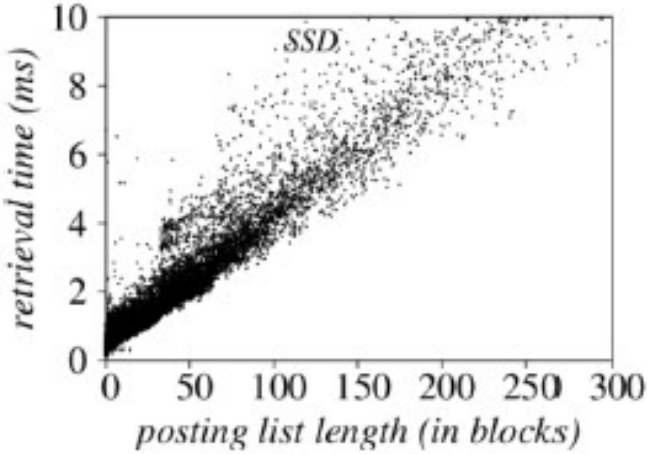
- Long lists have slower access time
  $\Longrightarrow$ Admitting too many short lists offers less benefit

How does SSD change the story? (Cont.)

Read access latency of posting lists of varying lengths



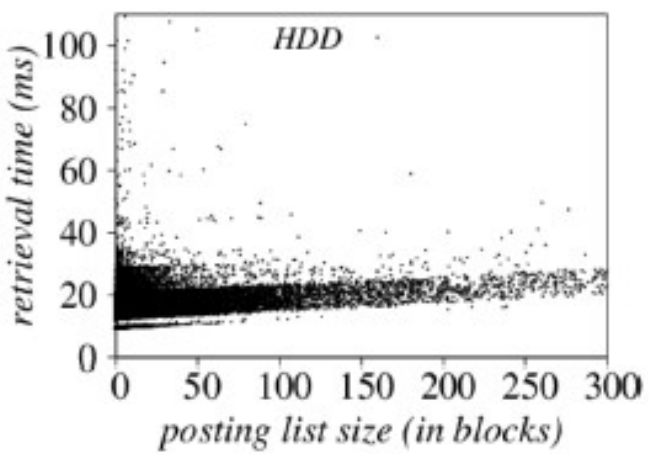
(a) on HDD (Seagate 3TB 7200rpm)



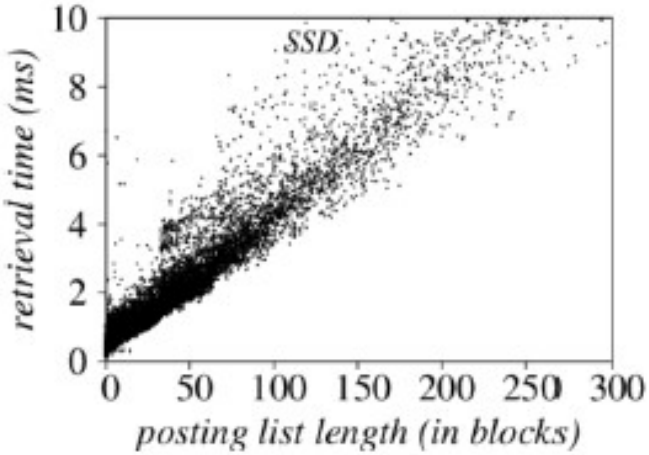
(b) on SSD (ADATA 256GB)

How does SSD change the story? (Cont.)

Read access latency of posting lists of varying lengths



(a) on HDD (Seagate 3TB 7200rpm)



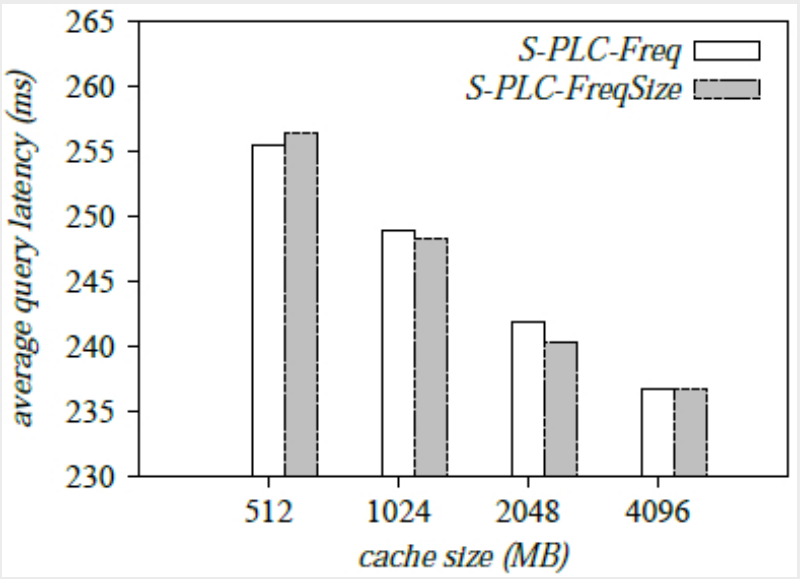
(b) on SSD (ADATA 256GB)

Conjecture 1

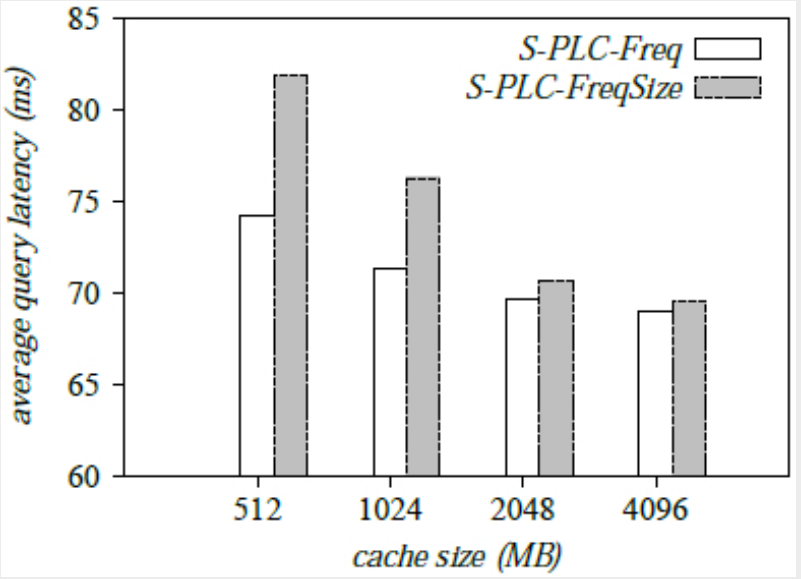
The best cache policy may have changed

Evaluation on Index Server

Static List Cache Policy



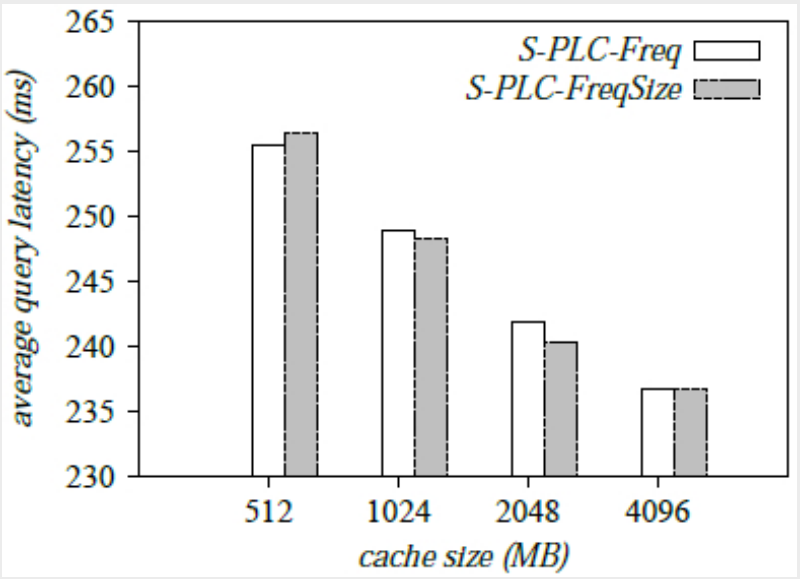
on HDD



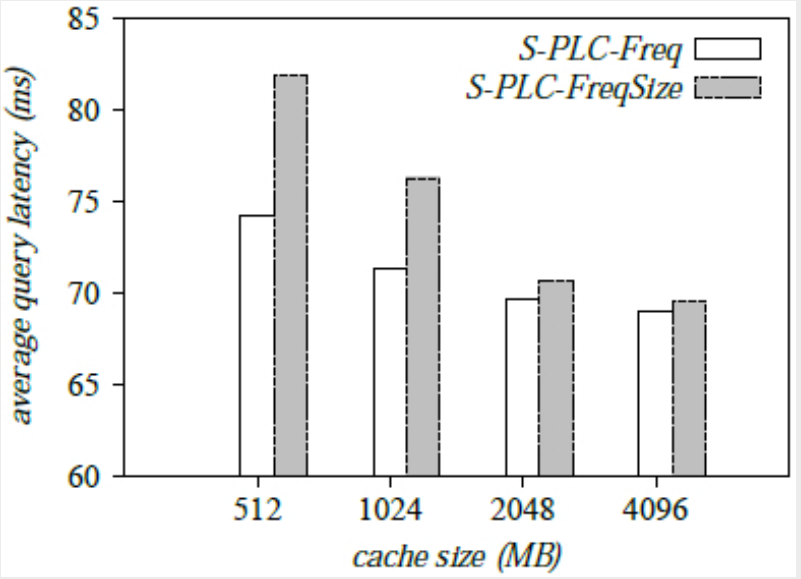
on SSD

Evaluation on Index Server

Static List Cache Policy



on HDD



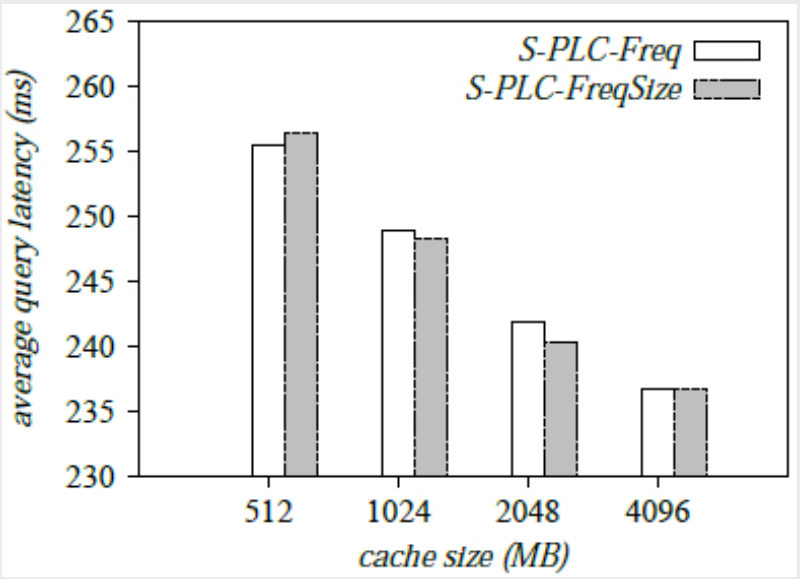
on SSD

Conjecture 1

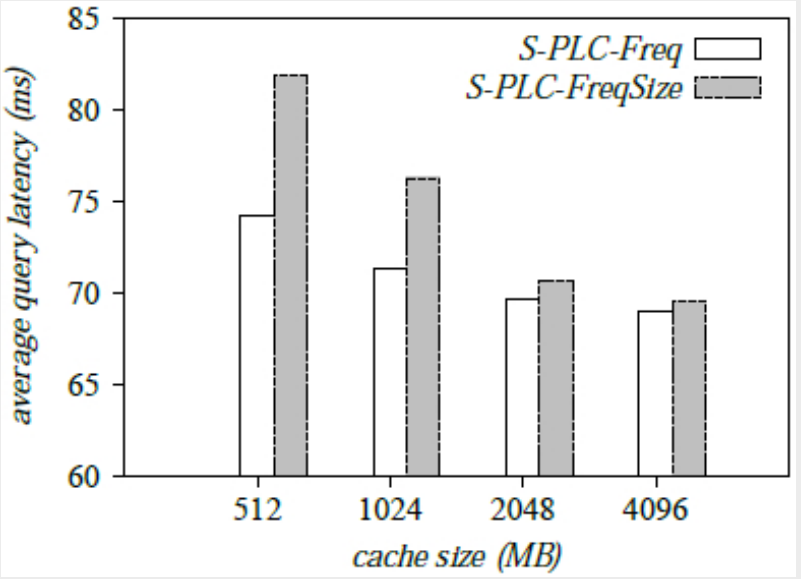
The best cache policy may have changed – Confirmed

Evaluation on Index Server

Static List Cache Policy



on HDD



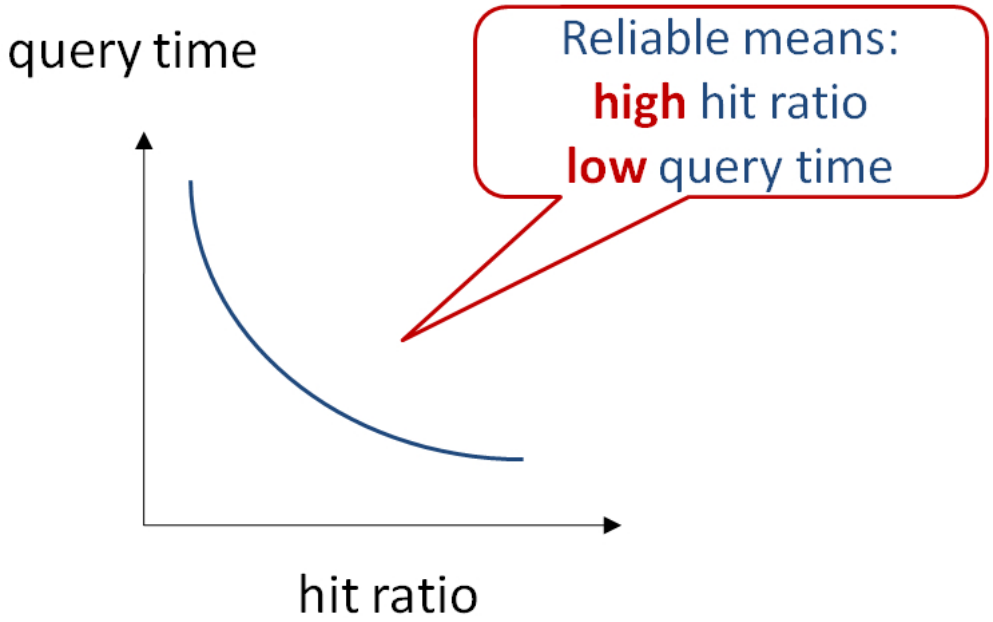
on SSD

Conjecture 1

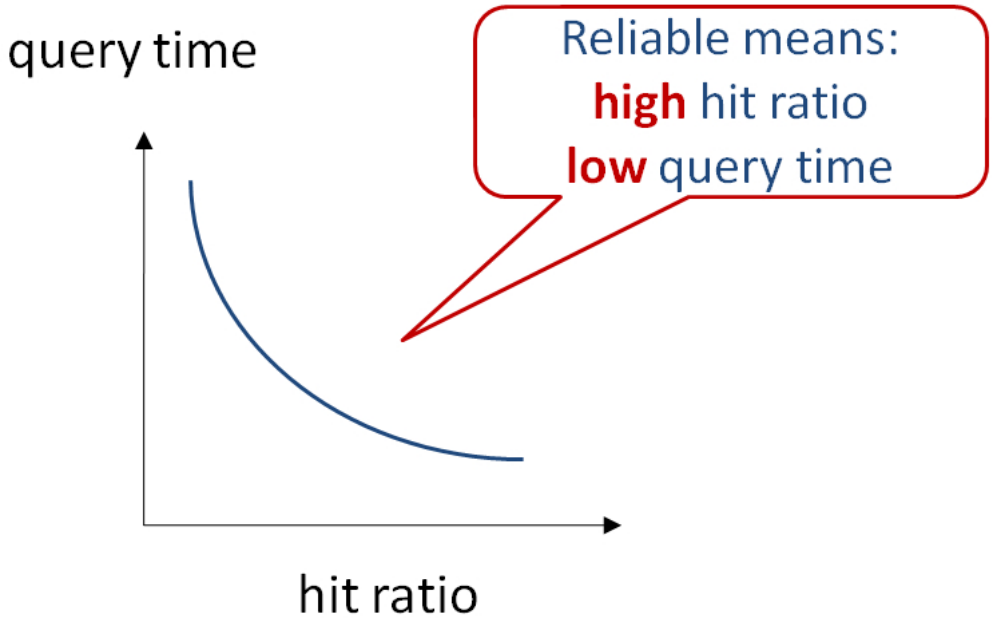
The best cache policy may have changed – Confirmed

★Do not rely on your knowledge acquired in HDD era.

Is Hit Ratio Reliable?

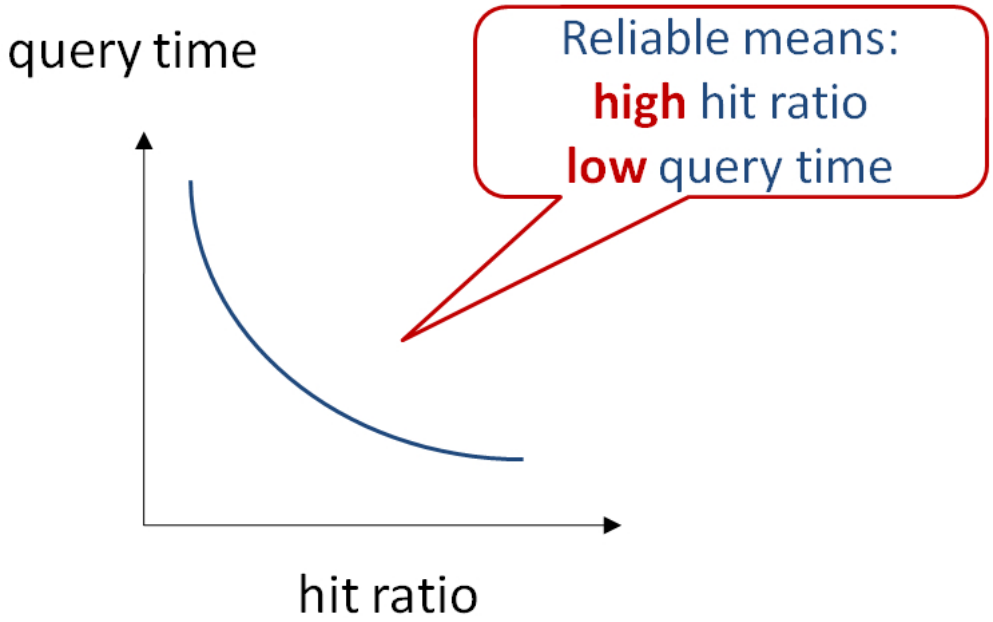


Is Hit Ratio Reliable?



Previously on HDD	Now on SSD
Reliable	?
As list access is a constant	NO
So, can measure the effectiveness of the cache	?

Is Hit Ratio Reliable?



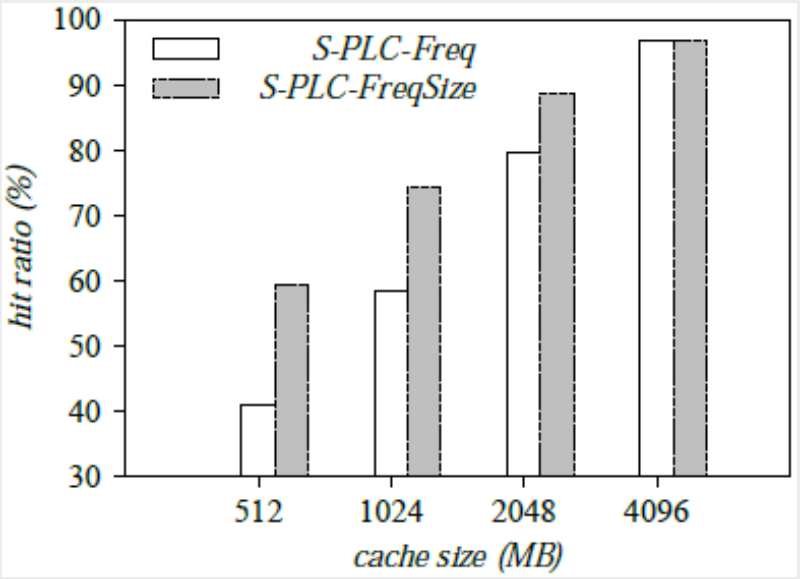
Previously on HDD	Now on SSD
Reliable	?
As list access is a constant	NO
So, can measure the effectiveness of the cache	?

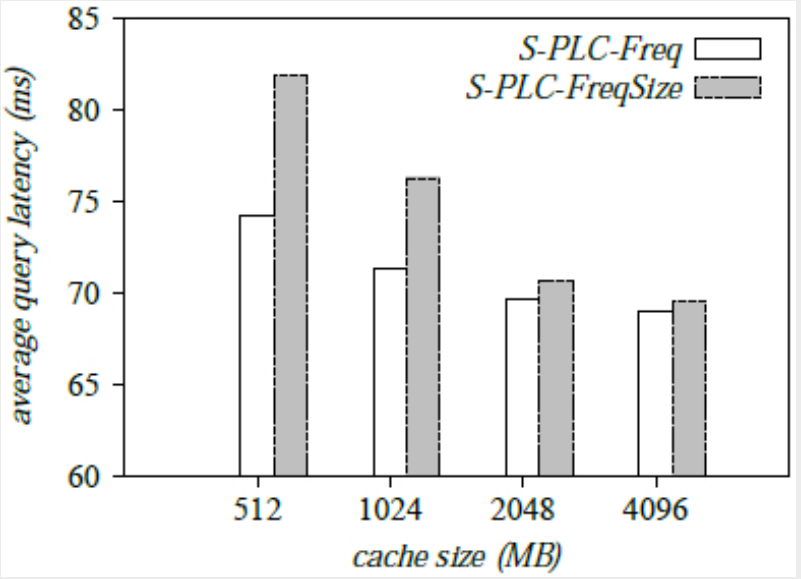
Conjecture 2

Cache hit ratio might not be reliable.

Is Hit Ratio Reliable? (Cont.)

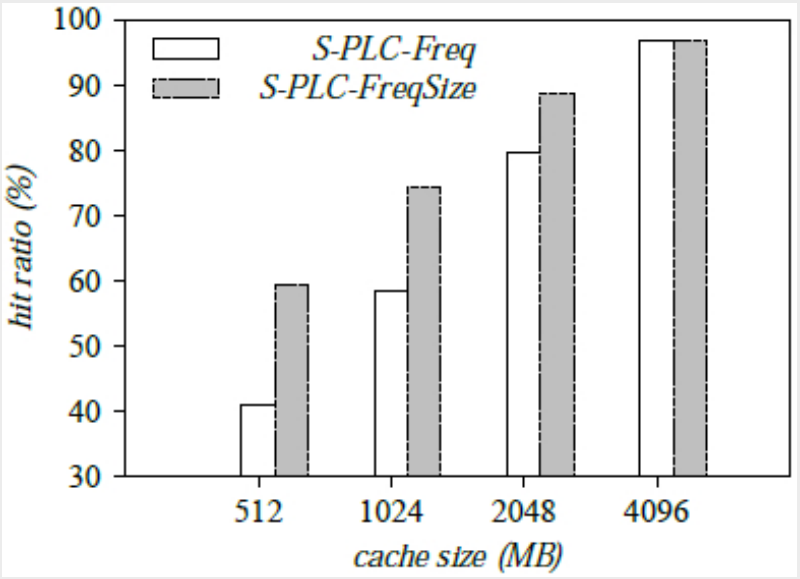
Static List Cache Policy (on SSD)

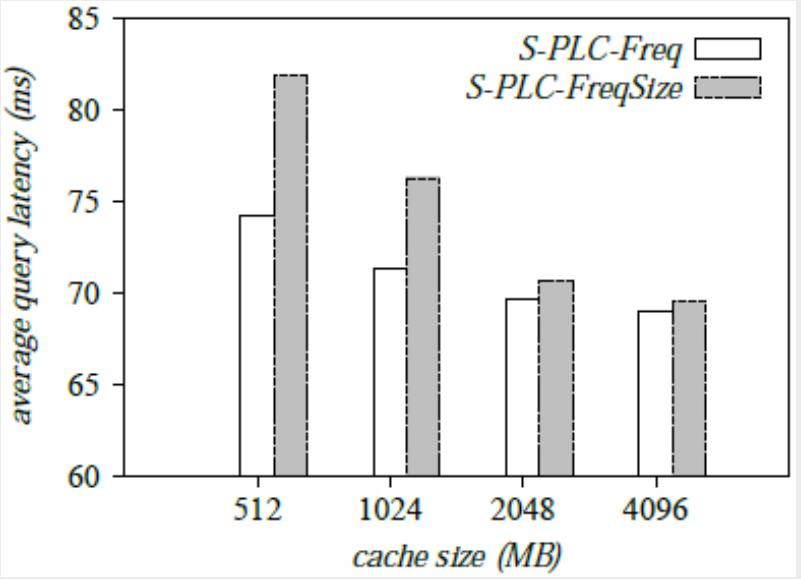




Is Hit Ratio Reliable? (Cont.)

Static List Cache Policy (on SSD)



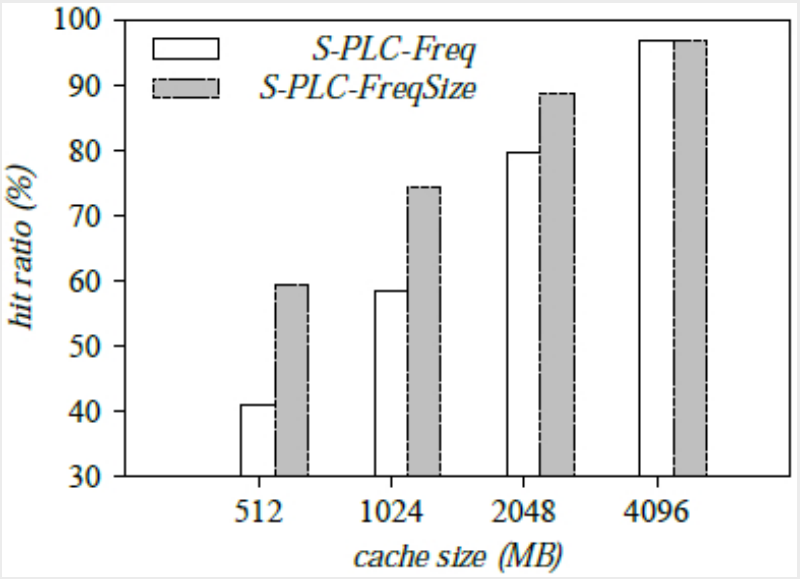


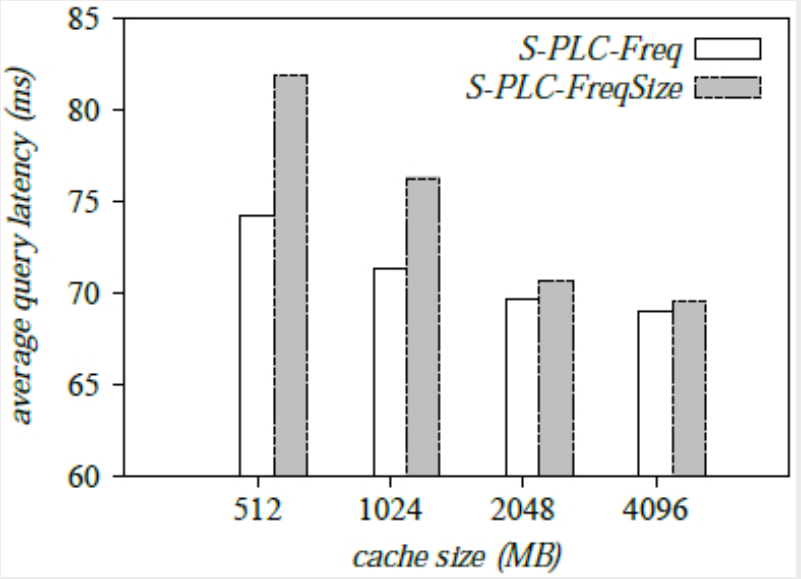
Conjecture 2

Cache hit ratio might not be reliable – Confirmed

Is Hit Ratio Reliable? (Cont.)

Static List Cache Policy (on SSD)



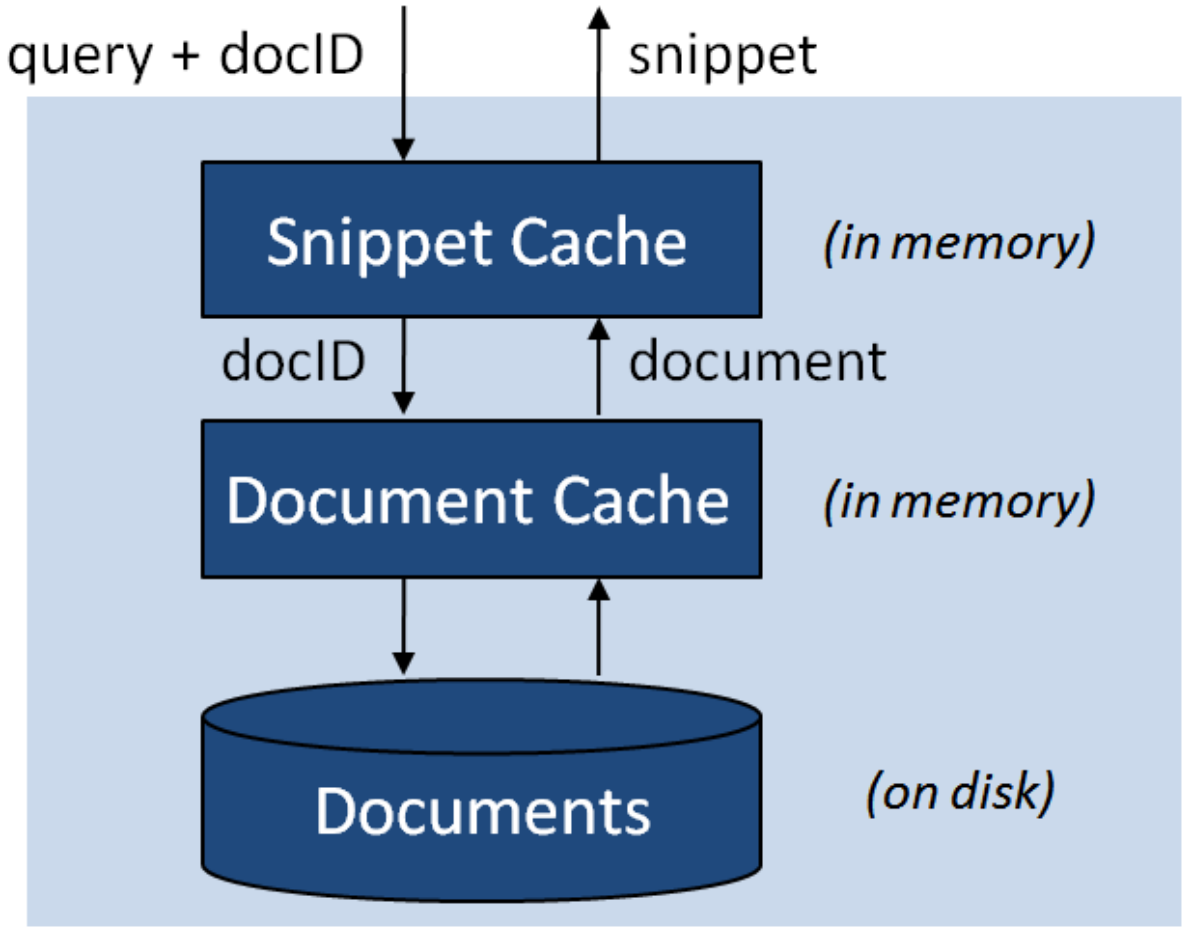


Conjecture 2

Cache hit ratio might not be reliable – Confirmed

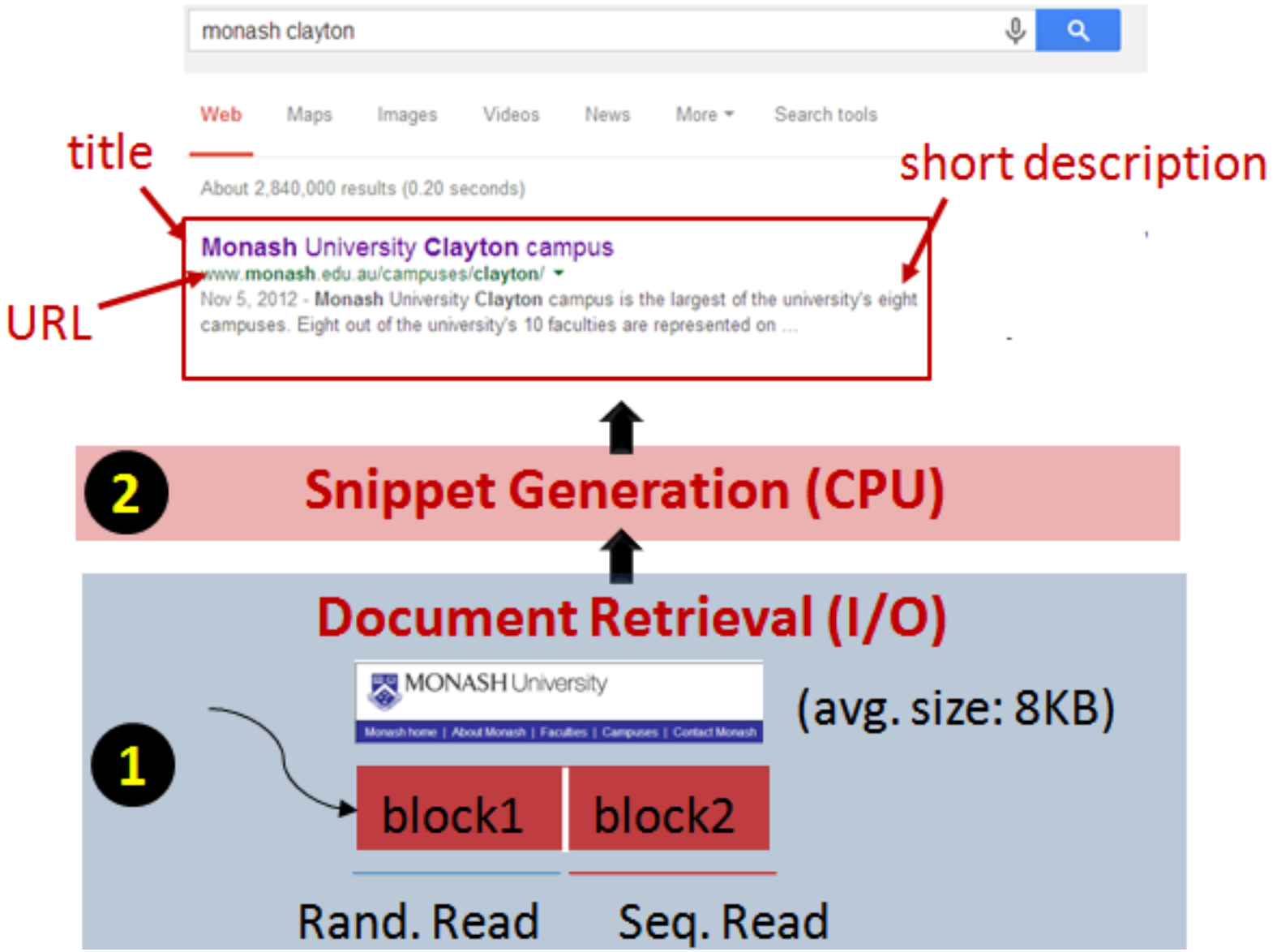
★Use query time, not hit ratio

Document Server and Its Cache

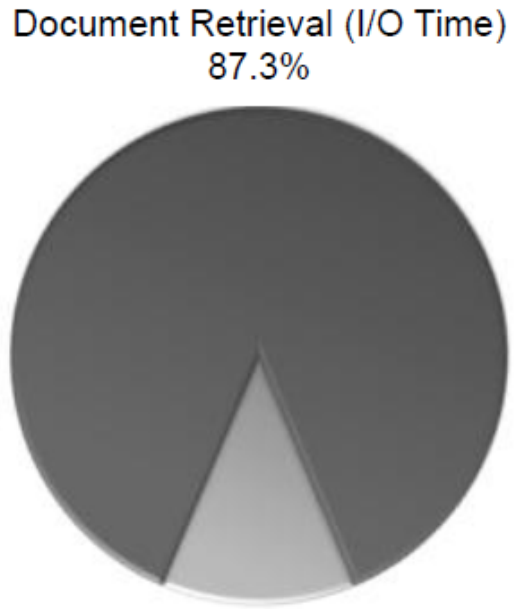


Document Server

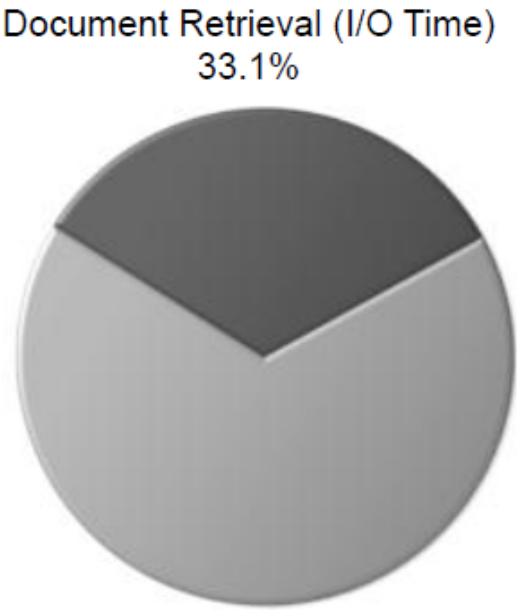
Document Server and Its Cache (Cont.)



Evaluation on Document Server

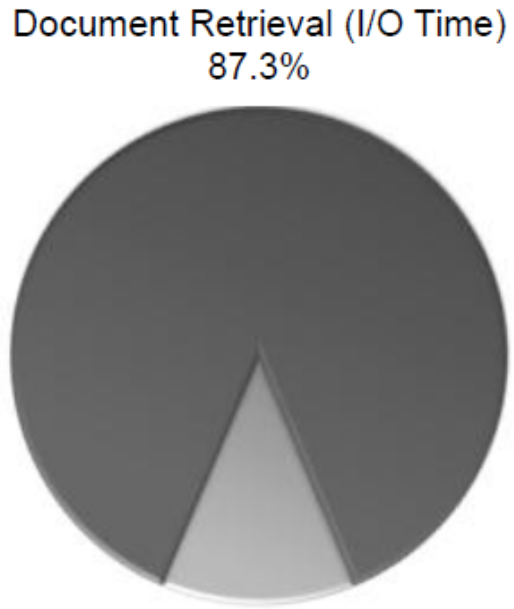


(a) on HDD

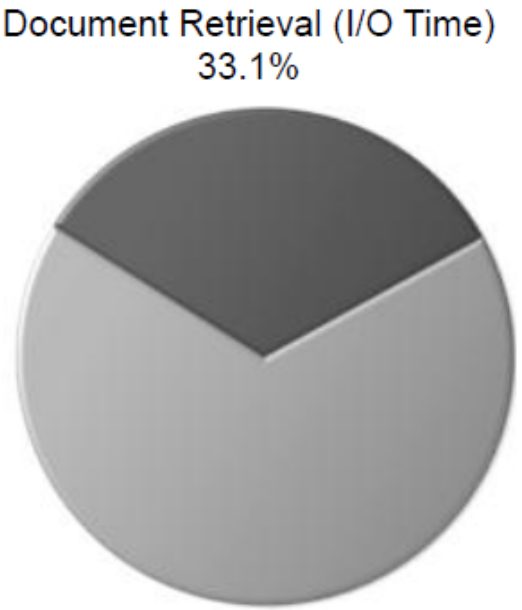


(b) on SSD

Evaluation on Document Server



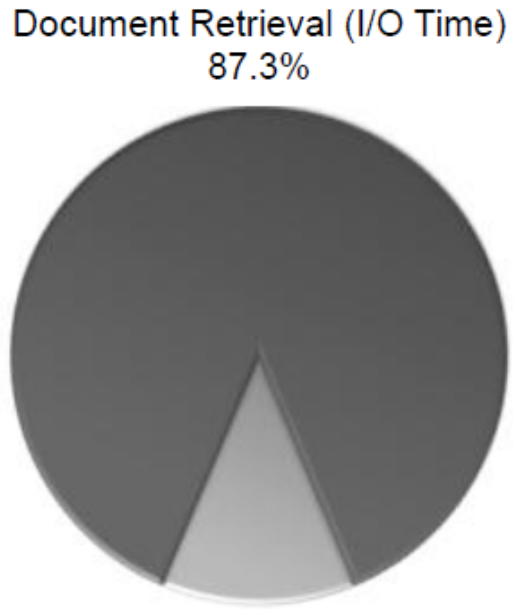
(a) on HDD



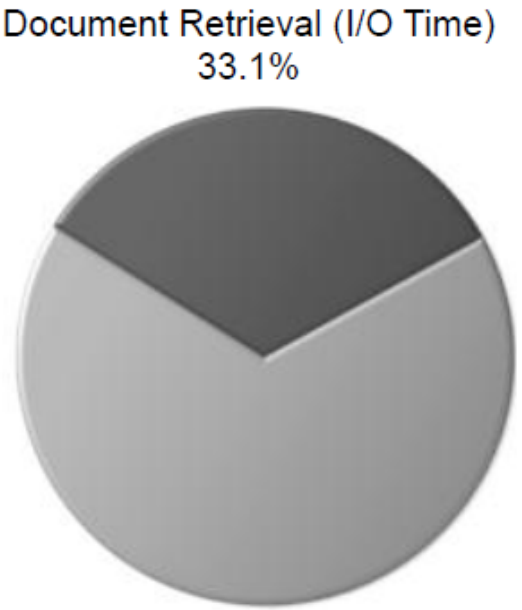
(b) on SSD

Document retrieval is no longer the bottleneck on document server

Evaluation on Document Server



(a) on HDD



(b) on SSD

Document retrieval is no longer the bottleneck on document server

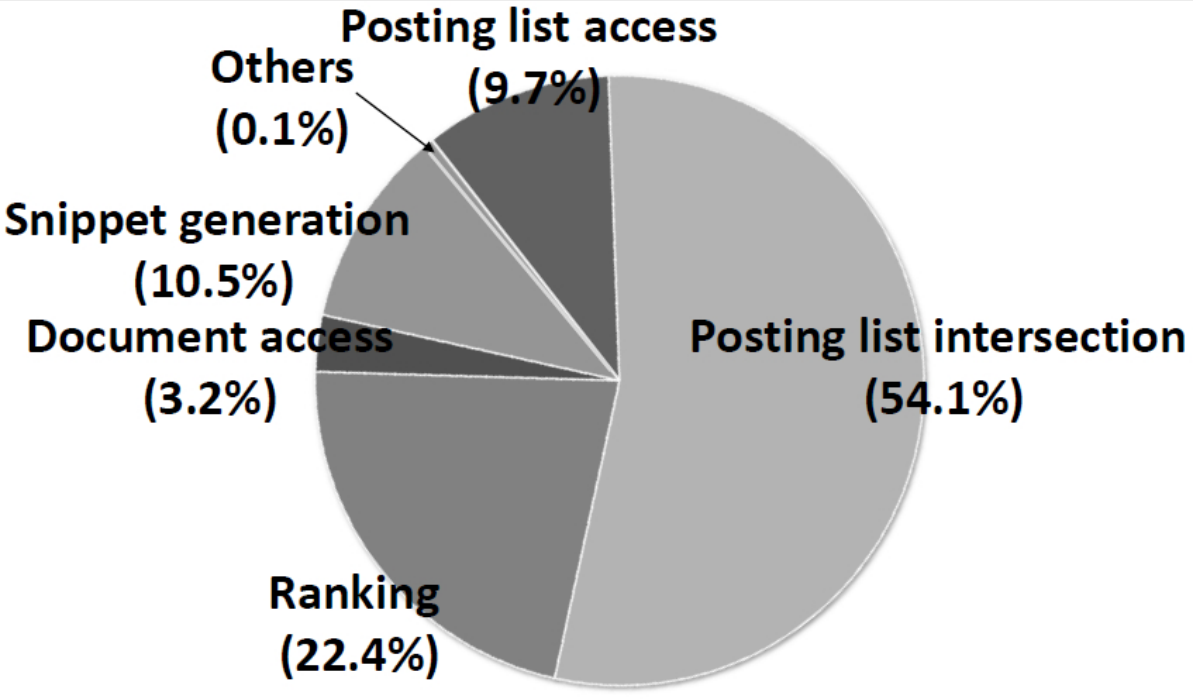
Conjecture 3

Is it the same case for the whole system?

The Impact on the Entire System Efficiency

Time break down for query processing (on SSD)

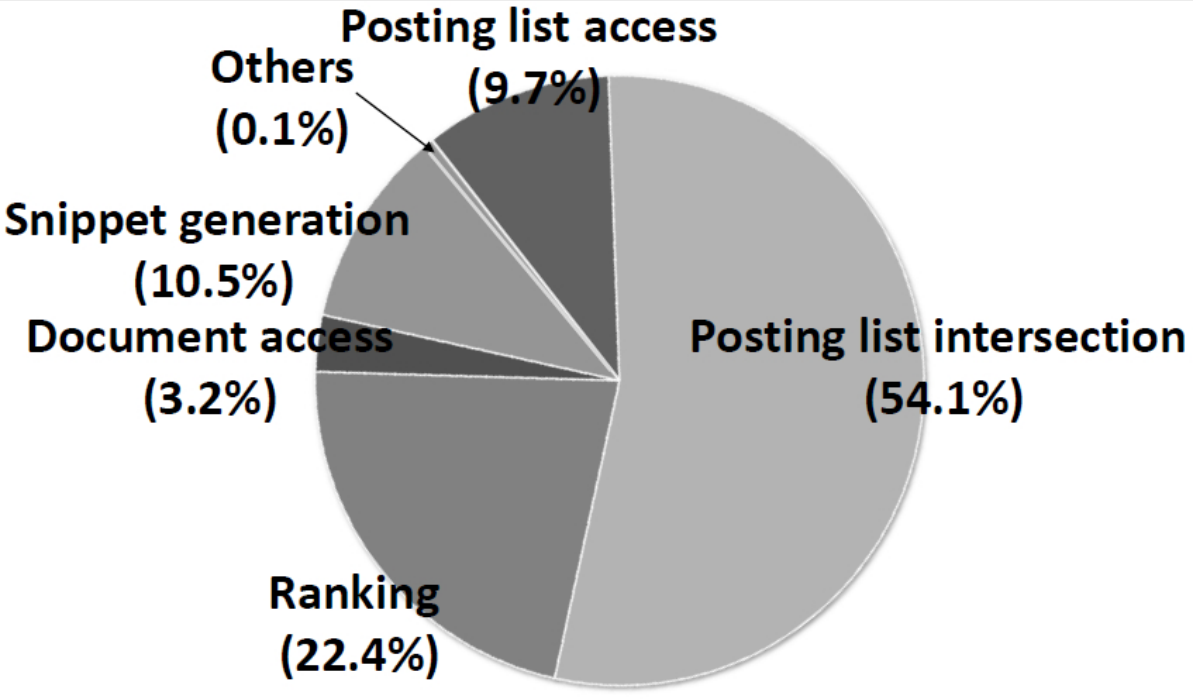
All caches are enabled



The Impact on the Entire System Efficiency

Time break down for query processing (on SSD)

All caches are enabled



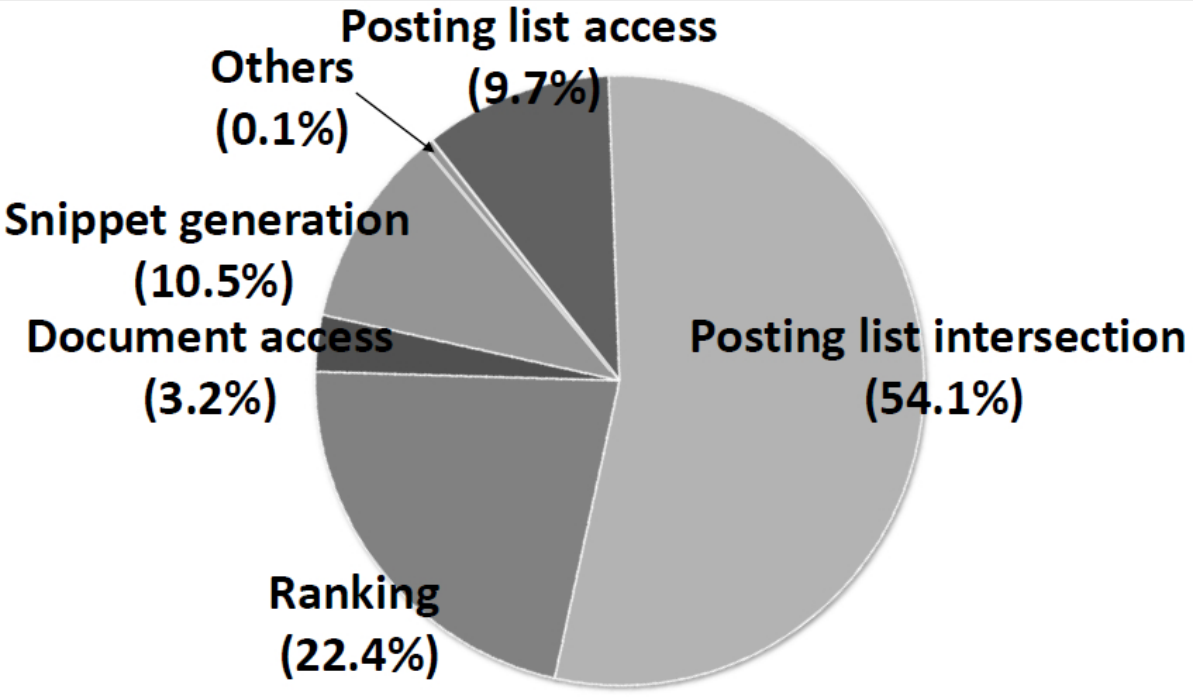
Conjecture 3

Is it the same case for the whole system? – Confirmed

The Impact on the Entire System Efficiency

Time break down for query processing (on SSD)

All caches are enabled



Conjecture 3

Is it the same case for the whole system? – Confirmed

★Disk accessing is no longer the bottleneck!

1 Background and Motivation

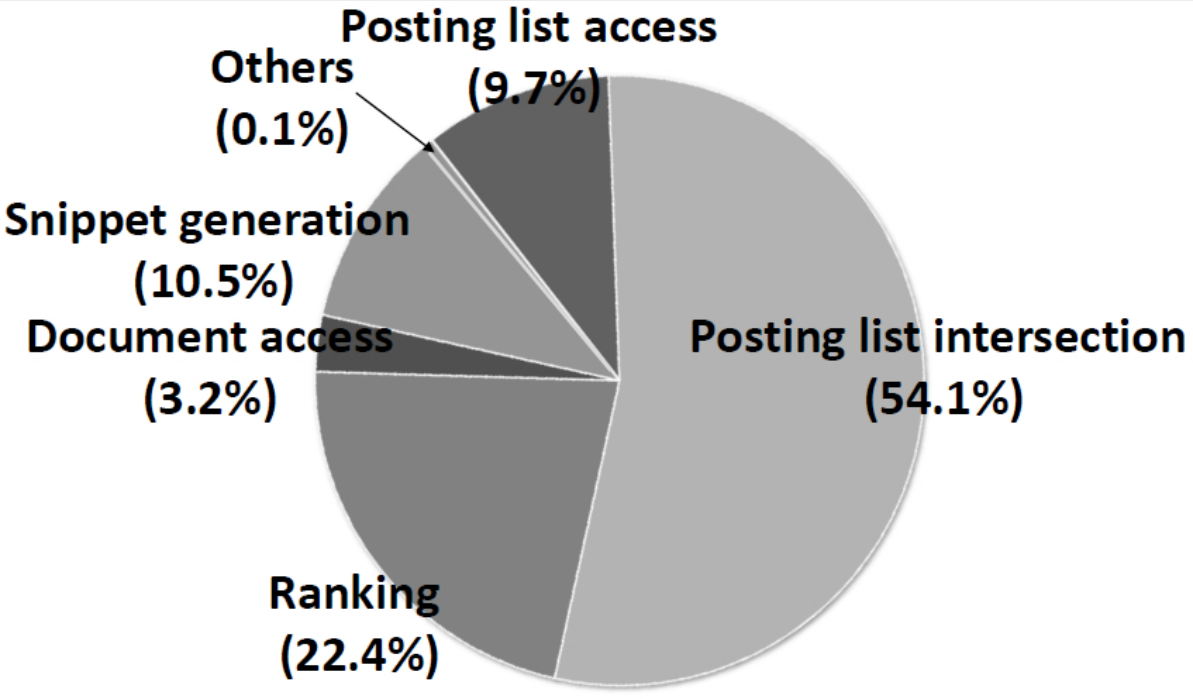
2 Research Questions and Answers

- RQ1: What is the impact of SSD on buffer management?
- RQ2: How could we deal with that?

The Impact on the Entire System Efficiency (Cont.)

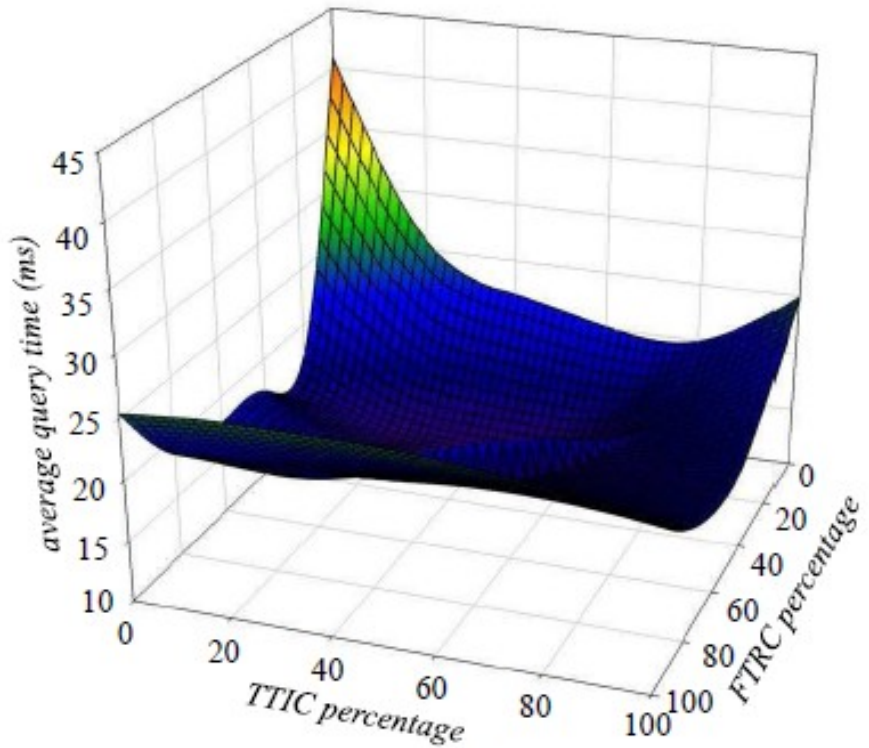
Time break down for query processing (on SSD)

All caches are enabled

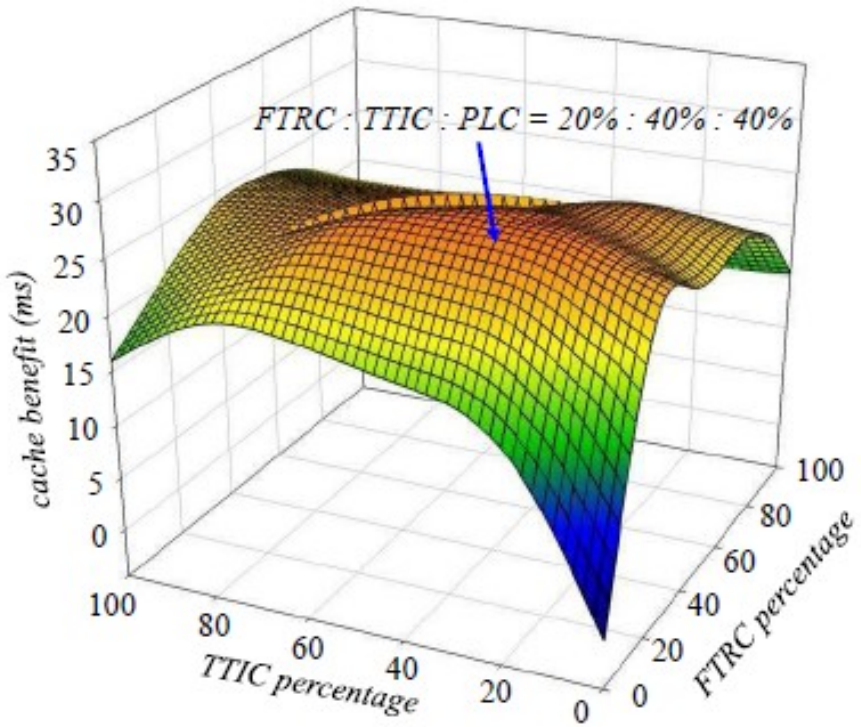


Problems	The time spent on intersection and ranking should be reduced
Solution	Adopting full-term-ranking-cache (FTRC) [Altingövde et al., 2011] and two-term-intersection-cache (TTIC) [Long and Suel, 2005]

Incorporating PLC+FTRC+TTIC

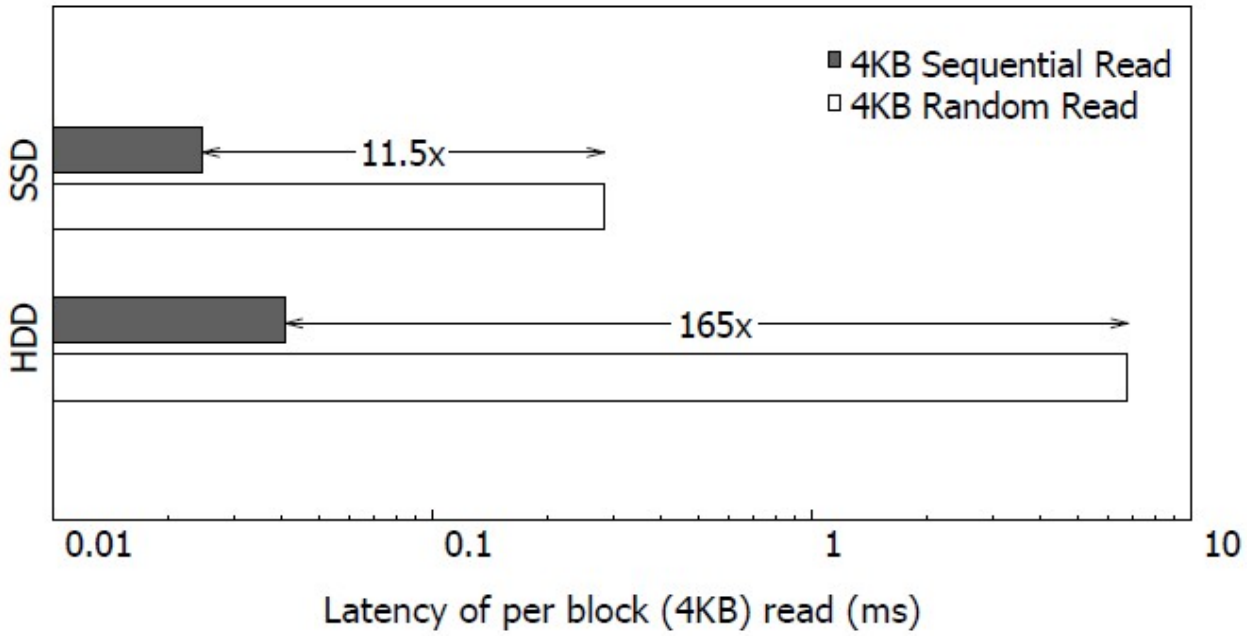


(a) query latency

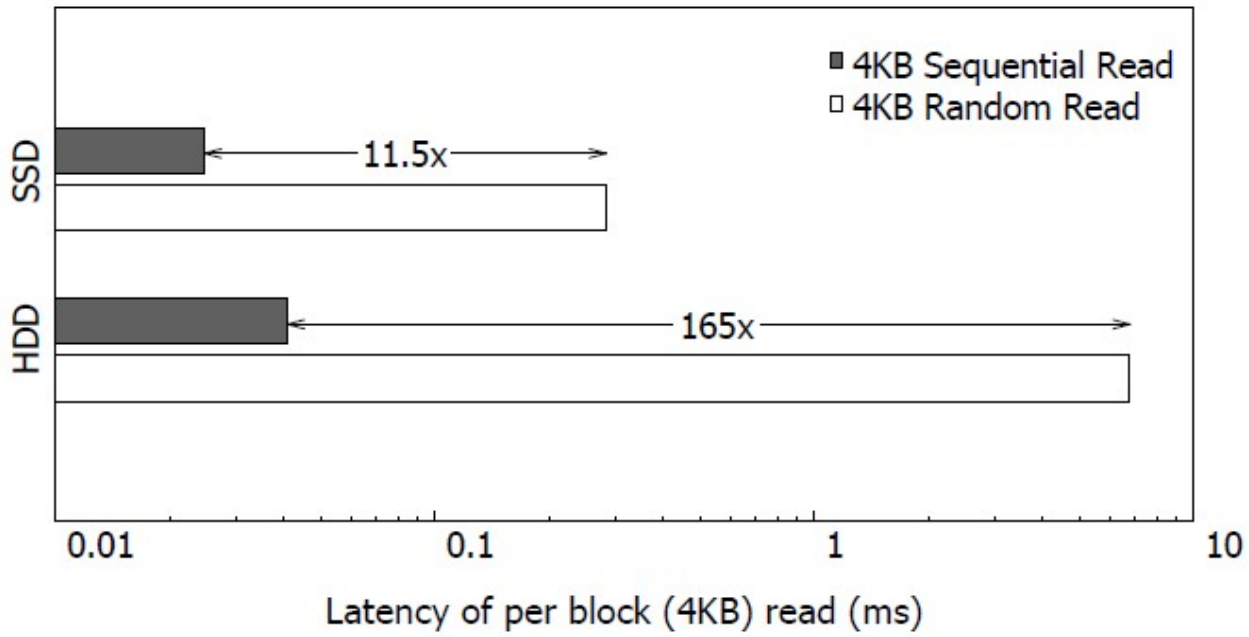


(b) cache benefit

New Static List Caching Policy



New Static List Caching Policy



Can we convert the latencies of random seek and sequential read into a uniform measure?

New Static List Caching Policy (Cont.)

Latency-Aware Caching (Static): BLOCK [Tong et al., 2013]

- For term t, $\mathcal{B}(t)$ represents the I/O cost that can be saved during the whole query evaluating process if $\ell(t)$ is kept in the cache
- Estimating model and existing static caching policies:

$$\mathcal{B}(t) = \mathcal{C}(t) \times \frac{f_q(t)}{|\ell(t)|} \tag{1}$$

$$\mathcal{B}(t) = \begin{cases} \frac{f_q(t)}{|\ell(t)|} & , \mathcal{C}(t) = 1 \quad \Leftrightarrow \quad \text{QTFDF} \\ f_q(t) & , \mathcal{C}(t) = |\ell(t)| \quad \Leftrightarrow \quad \text{QTF} \\ \frac{f_q(t)}{|\ell(t)|} \times \mathcal{T}(t) & , \mathcal{C}(t) = \mathcal{T}(t) \quad \Leftrightarrow \quad \text{MECH} \end{cases} \tag{2}$$

New Static List Caching Policy (Cont.)

Latency-Aware Caching (Static): BLOCK [Tong et al., 2013]

- For term t, $\mathcal{B}(t)$ represents the I/O cost that can be saved during the whole query evaluating process if $\ell(t)$ is kept in the cache
- Estimating model and existing static caching policies:

$$\mathcal{B}(t) = \mathcal{C}(t) \times \frac{f_q(t)}{|\ell(t)|} \tag{1}$$

$$\mathcal{B}(t) = \begin{cases} \frac{f_q(t)}{|\ell(t)|} & , \mathcal{C}(t) = 1 \quad \Leftrightarrow \quad \text{QTFDF} \\ f_q(t) & , \mathcal{C}(t) = |\ell(t)| \quad \Leftrightarrow \quad \text{QTF} \\ \frac{f_q(t)}{|\ell(t)|} \times \mathcal{T}(t) & , \mathcal{C}(t) = \mathcal{T}(t) \quad \Leftrightarrow \quad \text{MECH} \end{cases} \tag{2}$$

$$\mathcal{T}_{hdd}(t) = D_{seek} + D_{rotate} + D_{read} \times \frac{|\ell(t)|}{D_{block}} \tag{3}$$

$$\mathcal{T}_{ssd}(t) = S_{read} \times \frac{|\ell(t)|}{S_{page}} \tag{4}$$

New Static List Caching Policy (Cont.)

Latency-Aware Caching (Static): BLOCK [Tong et al., 2013]

- For term t, $\mathcal{B}(t)$ represents the I/O cost that can be saved during the whole query evaluating process if $\ell(t)$ is kept in the cache
- Estimating model and existing static caching policies:

$$\mathcal{B}(t) = \mathcal{C}(t) \times \frac{f_q(t)}{|\ell(t)|} \tag{1}$$

$$\mathcal{B}(t) = \begin{cases} \frac{f_q(t)}{|\ell(t)|} & , \mathcal{C}(t) = 1 \quad \Leftrightarrow \quad \text{QTFDF} \\ f_q(t) & , \mathcal{C}(t) = |\ell(t)| \quad \Leftrightarrow \quad \text{QTF} \\ \frac{f_q(t)}{|\ell(t)|} \times \mathcal{T}(t) & , \mathcal{C}(t) = \mathcal{T}(t) \quad \Leftrightarrow \quad \text{MECH} \end{cases} \tag{2}$$

$$\mathcal{T}_{hdd}(t) = D_{seek} + D_{rotate} + D_{read} \times \frac{|\ell(t)|}{D_{block}} \tag{3}$$

$$\mathcal{T}_{ssd}(t) = S_{read} \times \frac{|\ell(t)|}{S_{page}} \tag{4}$$

- New method: BLOCK

$$\mathcal{C}_{block}(t) = \overbrace{1}^{\text{1 random read}} + \underbrace{\left(\overbrace{\left\lceil \frac{|\ell(t)|}{B_s} \right\rceil}^{\text{\# of Sequential read}} - 1 \right) \div \gamma}_{\text{\# of equivalent random read}} \tag{5}$$

New Static List Caching Policy (Cont.)

Latency-Aware Caching (Static): BLOCK [Tong et al., 2013]

- For term t, $\mathcal{B}(t)$ represents the I/O cost that can be saved during the whole query evaluating process if $\ell(t)$ is kept in the cache
- Estimating model and existing static caching policies:

$$\mathcal{B}(t) = \mathcal{C}(t) \times \frac{f_q(t)}{|\ell(t)|} \tag{1}$$

$$\mathcal{B}(t) = \begin{cases} \frac{f_q(t)}{|\ell(t)|} & , \mathcal{C}(t) = 1 \quad \Leftrightarrow \quad \text{QTFDF} \\ f_q(t) & , \mathcal{C}(t) = |\ell(t)| \quad \Leftrightarrow \quad \text{QTF} \\ \frac{f_q(t)}{|\ell(t)|} \times \mathcal{T}(t) & , \mathcal{C}(t) = \mathcal{T}(t) \quad \Leftrightarrow \quad \text{MECH} \end{cases} \tag{2}$$

$$\mathcal{T}_{hdd}(t) = D_{seek} + D_{rotate} + D_{read} \times \frac{|\ell(t)|}{D_{block}} \tag{3}$$

$$\mathcal{T}_{ssd}(t) = S_{read} \times \frac{|\ell(t)|}{S_{page}} \tag{4}$$

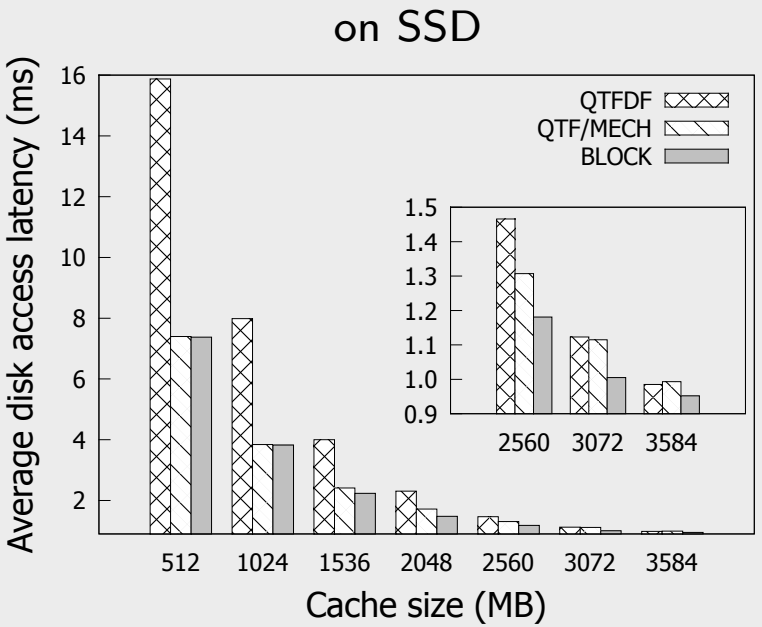
- New method: BLOCK

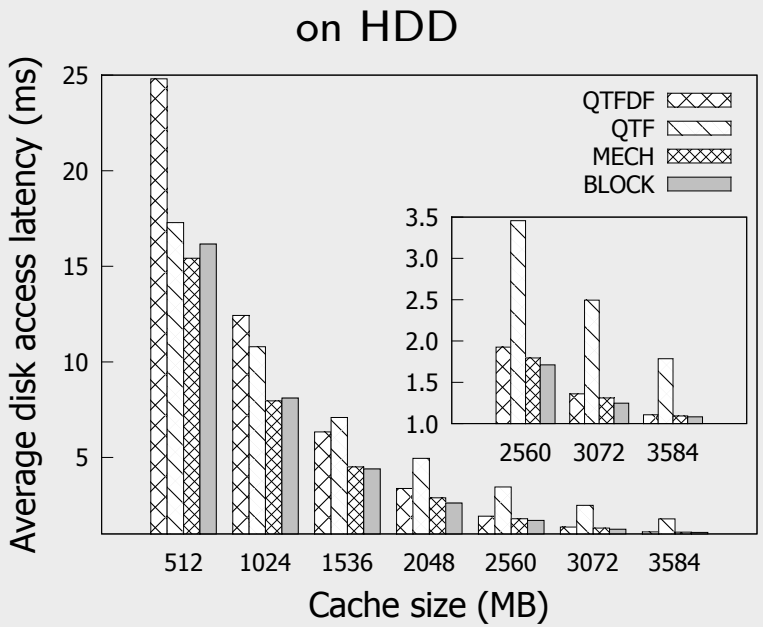
$$\mathcal{C}_{block}(t) = \overbrace{1}^{\text{1 random read}} + \underbrace{\left(\overbrace{\left\lceil \frac{|\ell(t)|}{B_s} \right\rceil}^{\text{\# of Sequential read}} - 1 \right) \div \gamma}_{\text{\# of equivalent random read}} \tag{5}$$

  - 1 access = 1 random seek + several sequential reads
    = 1 random seek + a few equivalent random seeks

New Static List Caching Policy (Cont.)

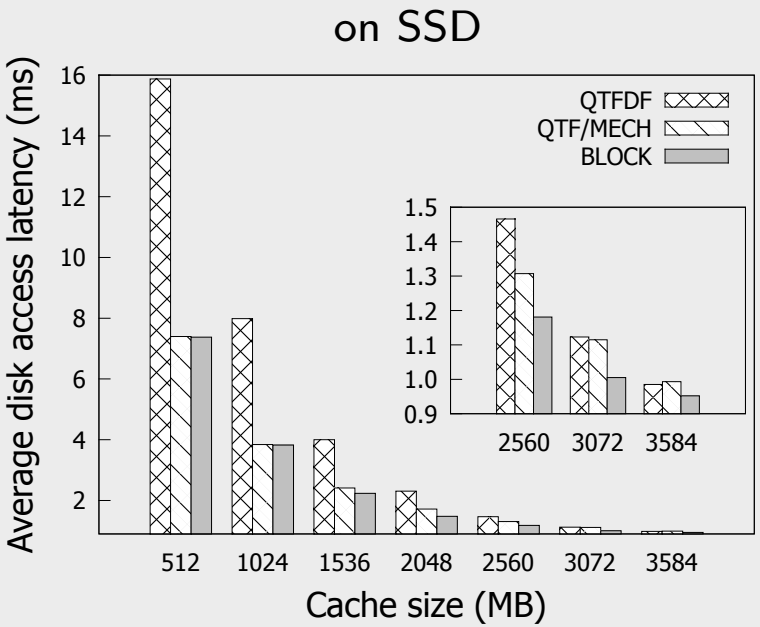
Experimental Results

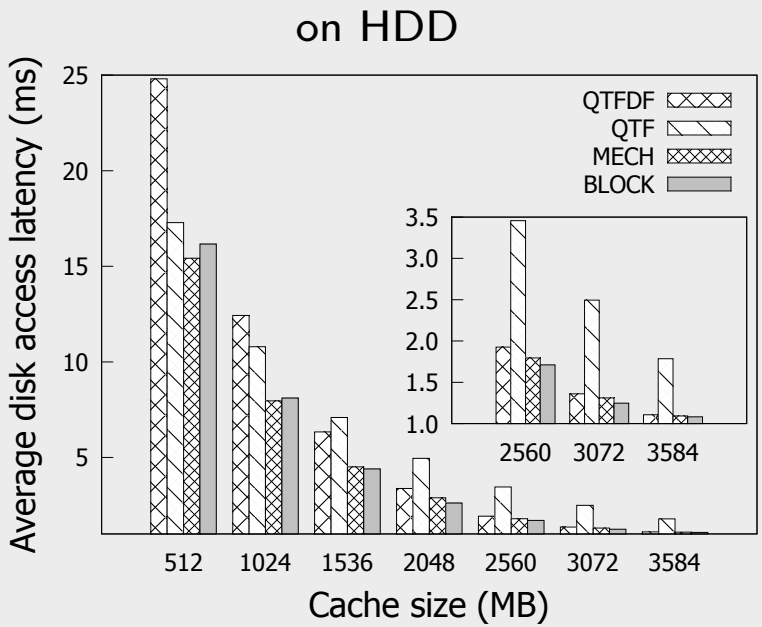




New Static List Caching Policy (Cont.)

Experimental Results

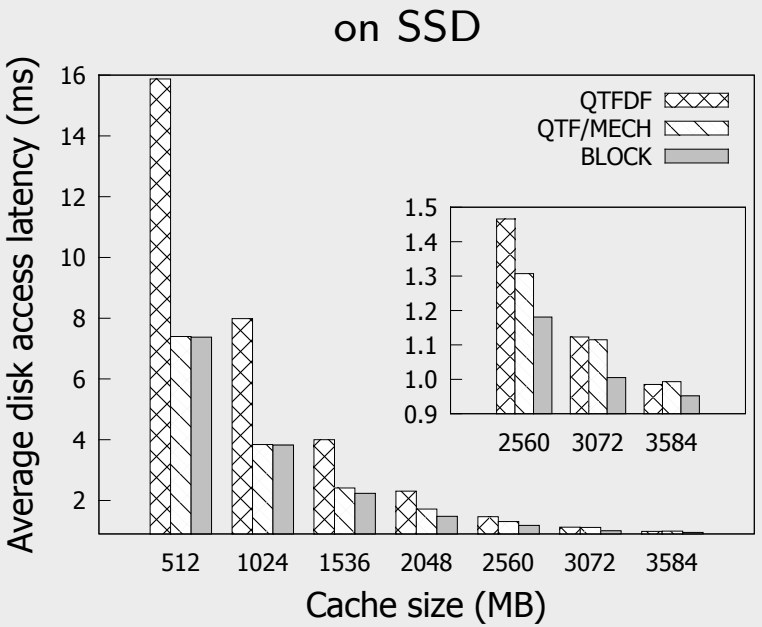


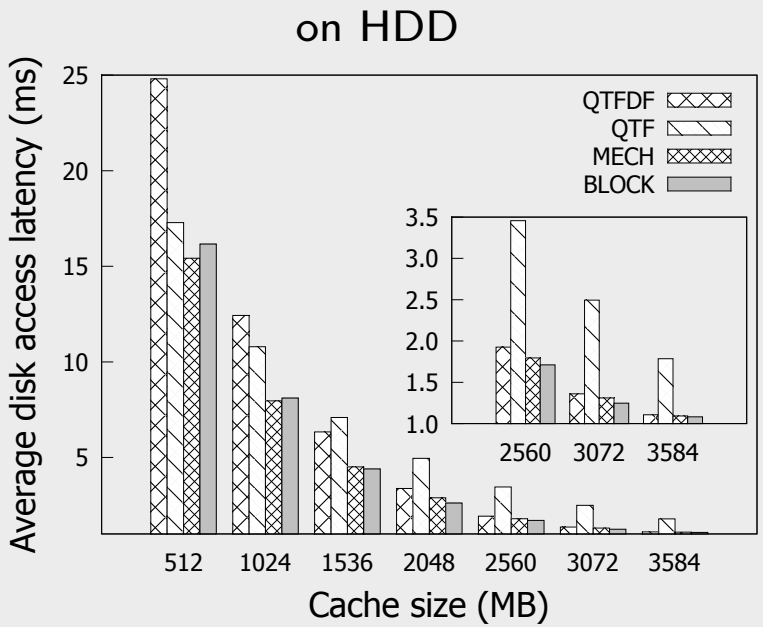


- BLOCK outperforms other methods on SSD

New Static List Caching Policy (Cont.)

Experimental Results

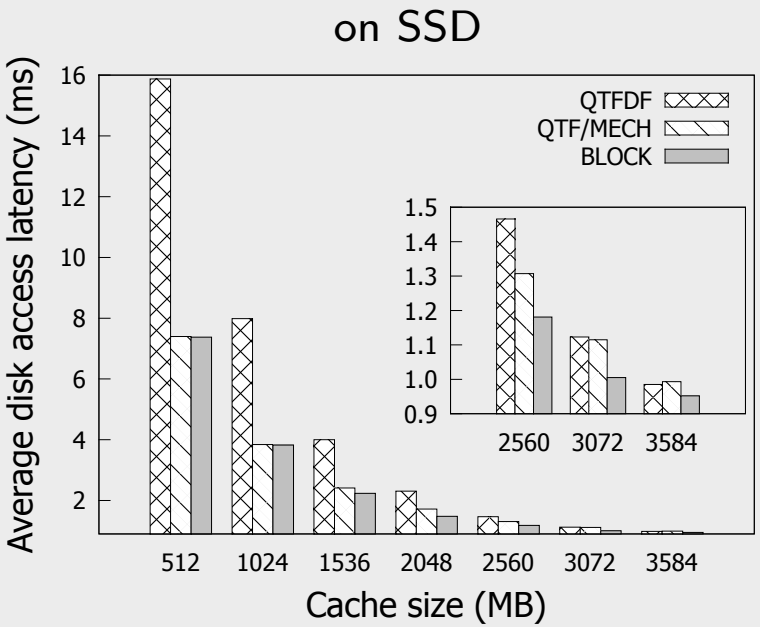


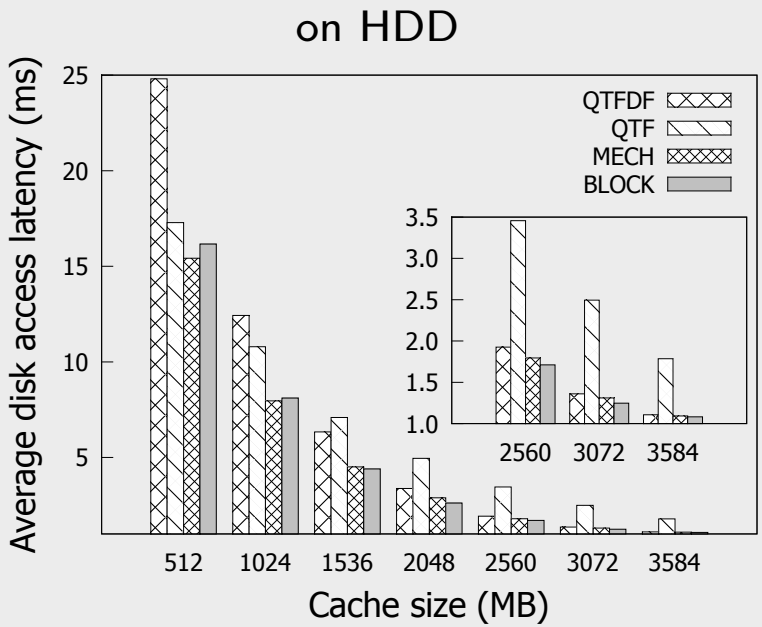


- BLOCK outperforms other methods on SSD
  1. BLOCK beats QTFDF (by far the best policy on HDD) by 20%-60%

New Static List Caching Policy (Cont.)

Experimental Results

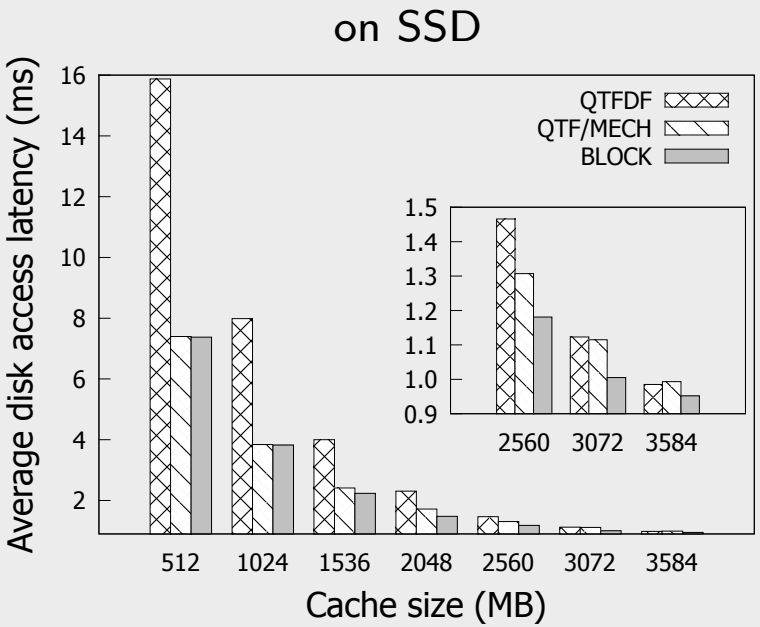


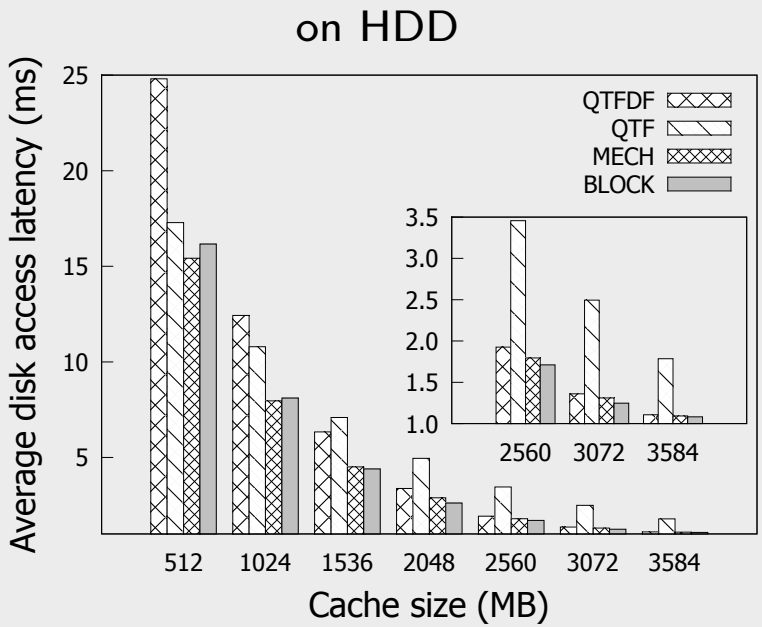


- BLOCK outperforms other methods on SSD
  1. BLOCK beats QTFDF (by far the best policy on HDD) by 20%-60%
  2. BLOCK outperforms QTF/MECH (identical on SSD) too

New Static List Caching Policy (Cont.)

Experimental Results





- BLOCK outperforms other methods on SSD
  1. BLOCK beats QTFDF (by far the best policy on HDD) by 20%-60%
  2. BLOCK outperforms QTF/MECH (identical on SSD) too
- BLOCK is better even on HDD (though not that significantly)

References


Altingövde, I. S., Ozcan, R., Cambazoglu, B. B., and Ulusoy, Ö. (2011).
Second chance: A hybrid approach for dynamic result caching in search engines.
In *ECIR*, pages 510–516.


Baeza-Yates, R. A., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., and Silvestri, F. (2007).
The impact of caching on search engines.
In *SIGIR*, pages 183–190.


Chen, F., Koufaty, D. A., and Zhang, X. (2009).
Understanding intrinsic characteristics and system implications of flash memory based solid state drives.
In *SIGMETRICS/Performance*, pages 181–192.


Garcia, J. M., Frohock, M., Reding, E., Reding, J. A., Garcia, M. F., DeLuca, S. A., and Whalen, E. (2000).
Microsoft SQL Server(tm) 2000 Administrator's Companion.
Microsoft Press.


Long, X. and Suel, T. (2005).
Three-level caching for efficient query processing in large web search engines.
In *WWW*, pages 257–266.


Tong, J., Wang, G., and Liu, X. (2013).
Latency-aware strategy for static list caching in flash-based web search engines.
In *CIKM*, pages 1209–1212.


Wang, J., Lo, E., Yiu, M. L., Tong, J., Wang, G., and Liu, X. (2013).
The impact of solid state drive on search engine cache management.
In *SIGIR*, pages 693–702.

Thanks for your time!

Questions & Comments?

For more details of RQ1, please refer to:

- J. Wang, E. Lo, M, Yiu, J. Tong, G. Wang, X. Liu,
  The impact of Solid State Disk on Search Engine Cache Management,
  In *SIGIR*, 2013, pp. 693-702.

For more details of RQ2, please refer to:

- J. Tong, G. Wang, X. Liu,
  Latency-Aware Strategy for Static List Caching in Flash-based Web Search Engines,
  In *CIKM*, 2013, pp. 1209-1212.