Enhancing recovery using an SSD buffer pool extension

Philipp Thau

University of Kaiserslautern

21-07-2014

Overview

What's going to be said

Overview

What's going to be said

- This presentation is about SSDs:
  - Using SSDs to speed up the database system
  - Involving recovery and speed it up as well

Overview

What's going to be said

- This presentation is about SSDs:
  - Using SSDs to speed up the database system
  - Involving recovery and speed it up as well
- In the introduction SSDs and some techniques to use them will be presented

Overview

What's going to be said

- This presentation is about SSDs:
  - Using SSDs to speed up the database system
  - Involving recovery and speed it up as well
- In the introduction SSDs and some techniques to use them will be presented
- After that I will focus on presenting an approach for using the SSD as a read cache ("SSD buffer pool extension")

Overview

What's going to be said

- This presentation is about SSDs:
  - Using SSDs to speed up the database system
  - Involving recovery and speed it up as well

- In the introduction SSDs and some techniques to use them will be presented
- After that I will focus on presenting an approach for using the SSD as a read cache ("SSD buffer pool extension")
- Based on that approach, recovery will be added and different methods of reusing a previously filled cache will be presented

Outline

Role of storage medium

The used storage medium has a great effect on the database system:

- Affects how the system interacts with the drive
- Affects how fast the system is
- Affects how much money the system costs
- Affects how reliable the system is

Role of storage medium

The used storage medium has a great effect on the database system:

- Affects how the system interacts with the drive
- Affects how fast the system is
- Affects how much money the system costs
- Affects how reliable the system is

Role of storage medium

The used storage medium has a great effect on the database system:

- Affects how the system interacts with the drive
- Affects how fast the system is
- Affects how much money the system costs
- Affects how reliable the system is

Role of storage medium

The used storage medium has a great effect on the database system:

- Affects how the system interacts with the drive
- Affects how fast the system is
- Affects how much money the system costs
- Affects how reliable the system is

Role of storage medium

The used storage medium has a great effect on the database system:

- Affects how the system interacts with the drive
- Affects how fast the system is
- Affects how much money the system costs
- Affects how reliable the system is

SSDs

Characteristics

In this work the focus was set on SSDs

SSDs are the most important storage medium next to hard drives:

- Store data permanent
- Have a high data density
- Are solid (technically and mechanical)
- Don't consume much electrical power

SSDs

Characteristics

In this work the focus was set on SSDs

SSDs are the most important storage medium next to hard drives:

- Store data permanent
- Have a high data density
- Are solid (technically and mechanical)
- Don't consume much electrical power

SSDs

Characteristics

In this work the focus was set on SSDs

SSDs are the most important storage medium next to hard drives:

- Store data permanent
- Have a high data density
- Are solid (technically and mechanical)
- Don't consume much electrical power

SSDs

Characteristics

In this work the focus was set on SSDs

SSDs are the most important storage medium next to hard drives:

- Store data permanent
- Have a high data density
- Are solid (technically and mechanical)
- Don't consume much electrical power

SSDs

Characteristics

In this work the focus was set on SSDs

SSDs are the most important storage medium next to hard drives:

- Store data permanent
- Have a high data density
- Are solid (technically and mechanical)
- Don't consume much electrical power

SSDs

Characteristics

In this work the focus was set on SSDs

SSDs are the most important storage medium next to hard drives:

- Store data permanent
- Have a high data density
- Are solid (technically and mechanical)
- Don't consume much electrical power

SSDs

Advantage

The most important benefit is the faster access time in terms of IOPS (Input/Output operations per second)

- Modern SSDs reach up to 100.000 IOPS
- Modern HDDs reach up to 200 IOPS
  - They can't spin faster than a natural limit

SSDs

Advantage

The most important benefit is the faster access time in terms of IOPS (Input/Output operations per second)

- Modern SSDs reach up to 100.000 IOPS
- Modern HDDs reach up to 200 IOPS
  - They can't spin faster than a natural limit

SSDs

Advantage

The most important benefit is the faster access time in terms of IOPS (Input/Output operations per second)

- Modern SSDs reach up to 100.000 IOPS
- Modern HDDs reach up to 200 IOPS
  - They can't spin faster than a natural limit

SSDs

Disadvantage

The most important drawback are the costs

For 90 euro you can get ...

- a modern SSD with 256 gigabyte
- or a modern HDD with 3 terabyte

SSDs

Disadvantage

The most important drawback are the costs

For 90 euro you can get ...

- a modern SSD with 256 gigabyte
- or a modern HDD with 3 terabyte

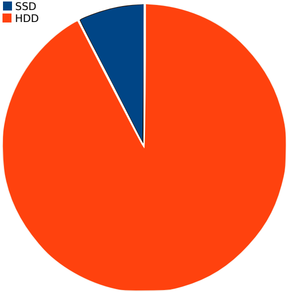
SSDs

Disadvantage

The most important drawback are the costs

For 90 euro you can get ...

- a modern SSD with 256 gigabyte
- or a modern HDD with 3 terabyte



SSDs vs. HDDs

Final comparison

SSDs

- up to 100.000 IOPS
  - around **1000** $\frac{\text{IOPS}}{\text{Euro}}$
- around 0.35 $\frac{\text{Euro}}{\text{Gigabyte}}$

HDDs

- up to 200 IOPS
  - around 2 $\frac{\text{IOPS}}{\text{Euro}}$
- around **0.03** $\frac{\text{Euro}}{\text{Gigabyte}}$

Both have their strengths!

SSDs vs. HDDs

Final comparison

SSDs

- up to 100.000 IOPS
  - around **1000** $\frac{\textbf{IOPS}}{\textbf{Euro}}$
- around 0.35 $\frac{\text{Euro}}{\text{Gigabyte}}$

HDDs

- up to 200 IOPS
  - around 2 $\frac{\text{IOPS}}{\text{Euro}}$
- around **0.03** $\frac{\textbf{Euro}}{\textbf{Gigabyte}}$

Both have their strengths!

SSDs vs. HDDs

Final comparison

SSDs

- up to 100.000 IOPS
  - around **1000** $\frac{\textbf{IOPS}}{\textbf{Euro}}$
- around 0.35 $\frac{\text{Euro}}{\text{Gigabyte}}$

HDDs

- up to 200 IOPS
  - around 2 $\frac{\text{IOPS}}{\text{Euro}}$
- around **0.03** $\frac{\textbf{Euro}}{\textbf{Gigabyte}}$

Both have their strengths!

SSDs vs. HDDs

Final comparison

SSDs

- up to 100.000 IOPS
  - around **1000** $\frac{\textbf{IOPS}}{\textbf{Euro}}$
- around 0.35 $\frac{\text{Euro}}{\text{Gigabyte}}$

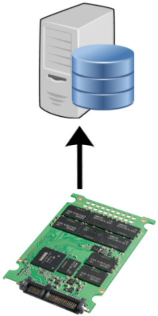
HDDs

- up to 200 IOPS
  - around 2 $\frac{\text{IOPS}}{\text{Euro}}$
- around **0.03** $\frac{\textbf{Euro}}{\textbf{Gigabyte}}$

Both have their strengths!

Entire database on SSD

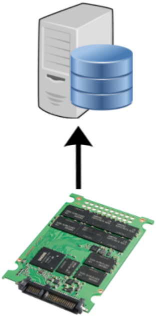
- Too expensive, as shown in the previous section!
- HDDs are better suited as basic storage medium



From now on: HDD on the lowest level in the storage hierarchy

Entire database on SSD

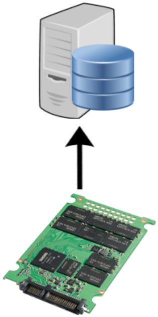
- Too expensive, as shown in the previous section!
- HDDs are better suited as basic storage medium



From now on: HDD on the lowest level in the storage hierarchy

Entire database on SSD

- Too expensive, as shown in the previous section!
- HDDs are better suited as basic storage medium

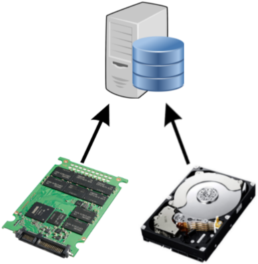


From now on: HDD on the lowest level in the storage hierarchy

SSD as disk on same level

System overview

- SSD is on the same level in the storage hierachy level as the HDD
- Both storage mediums serve the file requests



SSD as disk on same level

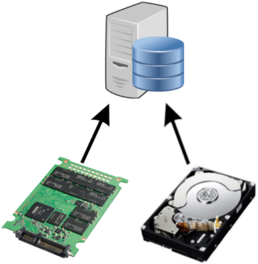
How it works

- SSD is just another storage medium on the same level as the HDD
- Indexes or the fact table of a data-warehouse may be saved there

SSD as disk on same level

How it works

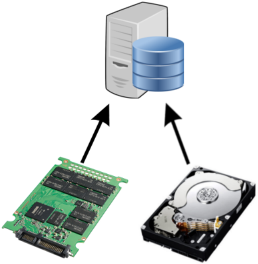
- SSD is just another storage medium on the same level as the HDD
- Indexes or the fact table of a data-warehouse may be saved there



SSD as disk on same level

Disadvantages

- Data has to be moved manually!
  - Admin may collect usage statistics on runtime and decides what to move then
- Data has to be moved again once the access pattern changes



SSD as disk on same level

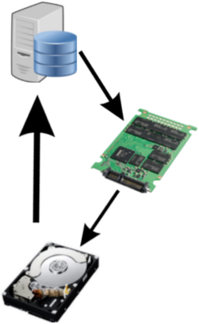
Disadvantages

- Data has to be moved manually!
    - Admin may collect usage statistics on runtime and decides what to move then
- Data has to be moved again once the access pattern changes

Write-back-cache

System overview

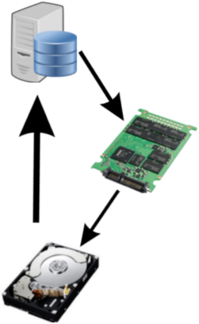
- SSD is on the level above the HDD
- HDD serves reads
- SSD supports writes



Write-back-cache

How it works

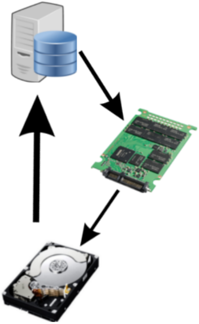
- Writes are buffered with the SSD
- With the help of the SSD the writes get transformed into sequential writes
  - When there is less load on the system, the writes are handed over to the HDD
- Exploits the strength of SSDs in terms of random IO
- Exploits the strength of HDDs in terms of sequential IO



Write-back-cache

How it works

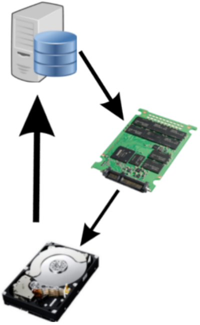
- Writes are buffered with the SSD
- With the help of the SSD the writes get transformed into sequential writes
  - When there is less load on the system, the writes are handed over to the HDD
- Exploits the strength of SSDs in terms of random IO
- Exploits the strength of HDDs in terms of sequential IO



Write-back-cache

How it works

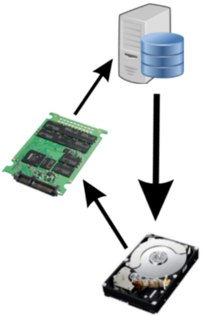
- Writes are buffered with the SSD
- With the help of the SSD the writes get transformed into sequential writes
  - When there is less load on the system, the writes are handed over to the HDD
- Exploits the strength of SSDs in terms of random IO
- Exploits the strength of HDDs in terms of sequential IO



SSD buffer pool extension ("SSD-Cache")

System overview

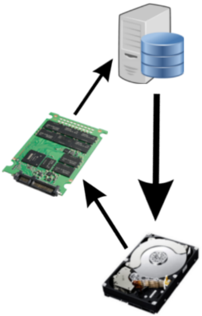
- SSD is on the level above the HDD
- Writes go primarily to the HDD
- SSD supports reads



SSD-Cache

How it works

- SSD acts as a read cache
- Gets filled with data automatically
  - Access patterns are recognized
- Exploits the strength of SSDs in terms of access time
- Exploits the strength of HDDs in terms of cheap space

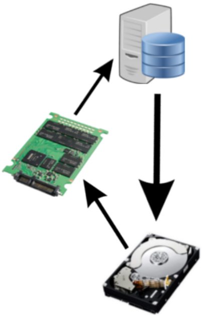


This approach combined with the recovery-aspect will be presented!

SSD-Cache

How it works

- SSD acts as a read cache
- Gets filled with data automatically
  - Access patterns are recognized
- Exploits the strength of SSDs in terms of access time
- Exploits the strength of HDDs in terms of cheap space

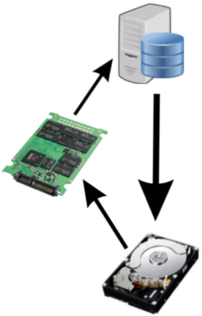


This approach combined with the recovery-aspect will be presented!

SSD-Cache

How it works

- SSD acts as a read cache
- Gets filled with data automatically
  - Access patterns are recognized
- Exploits the strength of SSDs in terms of access time
- Exploits the strength of HDDs in terms of cheap space

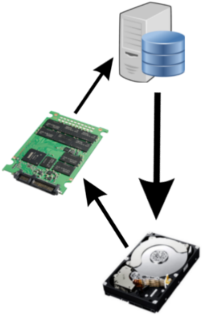


This approach combined with the recovery-aspect will be presented!

Outline

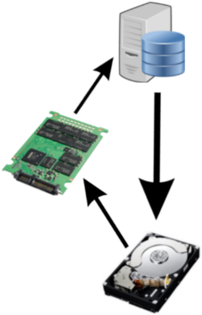
Overview

- An approach based on the "SSD-Cache"-concept will be presented
- It is designed as a write-through-cache (no buffering for writes)
- It uses a temperature based page replacement algorithm ("TAC")



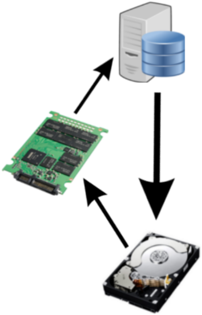
Overview

- An approach based on the "SSD-Cache"-concept will be presented
- It is designed as a write-through-cache (no buffering for writes)
- It uses a temperature based page replacement algorithm ("TAC")



Overview

- An approach based on the "SSD-Cache"-concept will be presented
- It is designed as a write-through-cache (no buffering for writes)
- It uses a temperature based page replacement algorithm ("TAC")

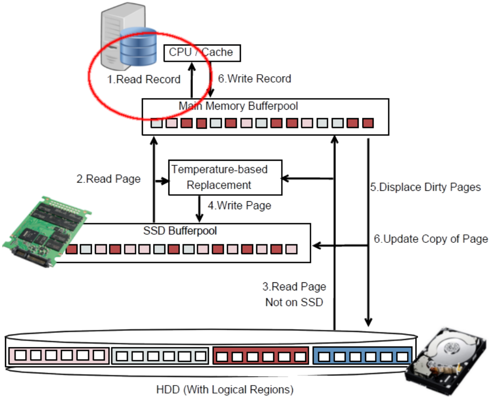


System setup

How the system works

1st step: Read record from main memory

- Check if the read can be served from main memory
- If the memory can serve the request: We are done
- If not: Go to step 2

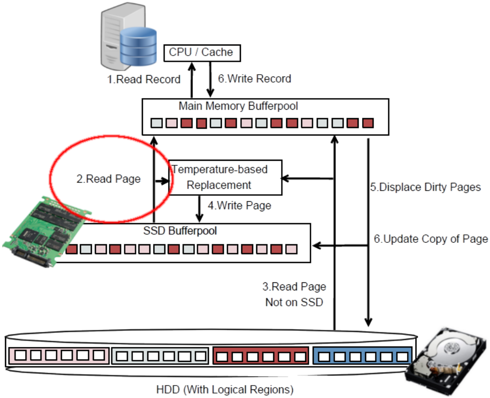


System setup

How the system works

2nd step: Read page from SSD

- Update the temperature of the region to read from
- Check if the read can be served from the SSD
- If the SSD can serve the request: We are done
- If not: Go to step 3

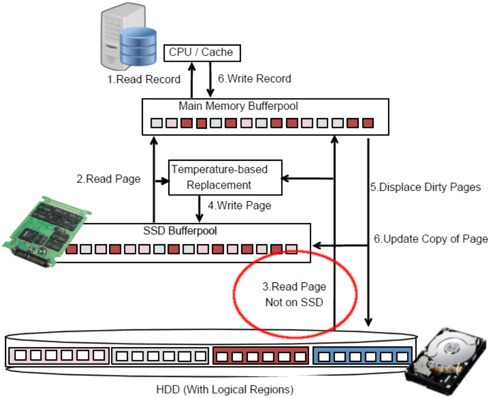


System setup

How the system works

3rd step: Read page from HDD

- All data is always present on the HDD so the request will finally be served
- Besides, the read data gets passed on to the replacement algorithm

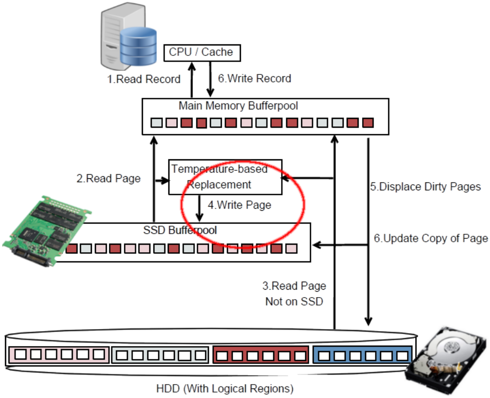


System setup

How the system works

4th step: Write page to SSD

- If the page gets accepted by the TAC-algorithm (meaning it is warmer than the coldest page cached) it will be saved to the SSD

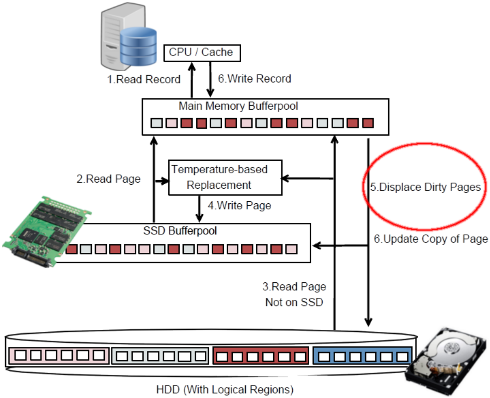


System setup

How the system works

5th step: Displace dirty pages

- When a dirty page gets displaced from the main memory: Go to step 6

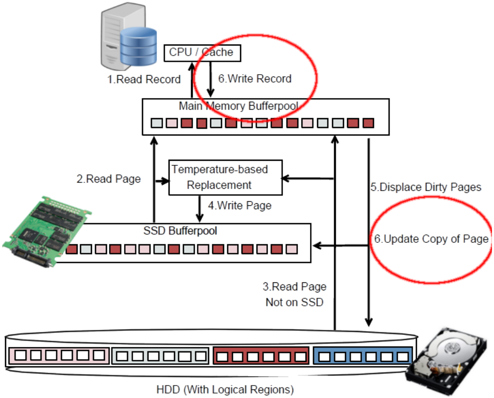


System setup

How the system works

6th step: Update copy of page

- The page gets updated on the HDD and (if present) in the SSD-Cache
- This means the pages in the SSD-Cache are always in the same state as their copy on the HDD



Temperature based replacement

How the algorithm works

Situation: Let $p_{\mathtt{new}}$ be a page read from the HDD (means: it is not present on the SSD)

- Case 1: SSD-Cache is not full:
  - Every page gets accepted
  - $p_{\mathtt{new}}$ is saved on the SSD
- Case 2: SSD-Cache is full:
  - If $p_{\mathtt{new}}$ is warmer than the coldest page $p_{\mathtt{cold}}$ on the SSD: $p_{\mathtt{new}}$ overwrites $p_{\mathtt{cold}}$
  - Else: $p_{\mathtt{new}}$ gets rejected

Temperature based replacement

How the algorithm works

Situation: Let $p_{\texttt{new}}$ be a page read from the HDD (means: it is not present on the SSD)

- Case 1: SSD-Cache is not full:
  - Every page gets accepted
  - $p_{\texttt{new}}$ is saved on the SSD
- Case 2: SSD-Cache is full:
  - If $p_{\texttt{new}}$ is warmer than the coldest page $p_{\texttt{cold}}$ on the SSD: $p_{\texttt{new}}$ overwrites $p_{\texttt{cold}}$
  - Else: $p_{\texttt{new}}$ gets rejected

Temperature based replacement

How the algorithm works

Situation: Let $p_{\mathtt{new}}$ be a page read from the HDD (means: it is not present on the SSD)

- Case 1: SSD-Cache is not full:
  - Every page gets accepted
  - $p_{\mathtt{new}}$ is saved on the SSD
- Case 2: SSD-Cache is full:
  - If $p_{\mathtt{new}}$ is warmer than the coldest page $p_{\mathtt{cold}}$ on the SSD: $p_{\mathtt{new}}$ overwrites $p_{\mathtt{cold}}$
  - Else: $p_{\mathtt{new}}$ gets rejected

Temperature based replacement

How the temperatures are calculated

- Temperatures get updated when pages are read from the HDD or the SSD
  - It is not relevant whether the page gets accepted or rejected by the replacement algorithm
  - This is how the system adapts to access patterns
- Temperatures are managed at the level of regions (e.g. 32 pages)
  - This makes the temperatures meaningful faster and uses less storage
- Random IO is prefered over sequential IO
  - To detect if the access type is random or sequential, a windowing technique is utilized
  - If at least 2 out of 20 reads go to the same region, those reads are declared as sequential IO

Temperature based replacement

How the temperatures are calculated

- Temperatures get updated when pages are read from the HDD or the SSD
  - It is not relevant whether the page gets accepted or rejected by the replacement algorithm
  - This is how the system adapts to access patterns
- Temperatures are managed at the level of regions (e.g. 32 pages)
  - This makes the temperatures meaningful faster and uses less storage
- Random IO is prefered over sequential IO
  - To detect if the access type is random or sequential, a windowing technique is utilized
  - If at least 2 out of 20 reads go to the same region, those reads are declared as sequential IO

Temperature based replacement

How the temperatures are calculated

- Temperatures get updated when pages are read from the HDD or the SSD
  - It is not relevant whether the page gets accepted or rejected by the replacement algorithm
  - This is how the system adapts to access patterns
- Temperatures are managed at the level of regions (e.g. 32 pages)
  - This makes the temperatures meaningful faster and uses less storage
- Random IO is prefered over sequential IO
  - To detect if the access type is random or sequential, a windowing technique is utilized
  - If at least 2 out of 20 reads go to the same region, those reads are declared as sequential IO

Temperature based replacement

Why not just use LRU?

- LRU: Replace the page that was not needed for the longest time
- TAC: Replace the page that has the lowest probability of being read

- With LRU, it may happen that a page which gets read only once replaces a page which soon will be needed
- LRU makes no difference between random IO and sequential IO

There will be some numbers shown in the “Results” section later

Temperature based replacement

Why not just use LRU?

- LRU: Replace the page that was not needed for the longest time
- TAC: Replace the page that has the lowest probability of being read

- With LRU, it may happen that a page which gets read only once replaces a page which soon will be needed
- LRU makes no difference between random IO and sequential IO

There will be some numbers shown in the "Results" section later

Temperature based replacement

Why not just use LRU?

- LRU: Replace the page that was not needed for the longest time
- TAC: Replace the page that has the lowest probability of being read

- With LRU, it may happen that a page which gets read only once replaces a page which soon will be needed
- LRU makes no difference between random IO and sequential IO

There will be some numbers shown in the "Results" section later

Temperature based replacement

Why not just use LRU?

- LRU: Replace the page that was not needed for the longest time
- TAC: Replace the page that has the lowest probability of being read

- With LRU, it may happen that a page which gets read only once replaces a page which soon will be needed
- LRU makes no difference between random IO and sequential IO

There will be some numbers shown in the "Results" section later

Metadata

- Metadata is saved in the main memory
- A hashtable is used to assign the position of the page on the HDD to the position of the page on the SSD, if the page is existing on the SSD
- The temperature statistics are also saved with a region based hashtable
- To identify the coldest page a heap is maintained:
    - When the coldest page needs to be identified, the temperature of the root of the heap gets updated 5 times
    - After that, the root is chosen as the coldest page
    - The root may not be the absolute coldest page after the 5 updates, but the time needed is still logarithmic ($O(\mathrm{n})$ otherwise with heapify()):

$$O(5 * \log(\mathrm{n})) = O(\log(\mathrm{n}))$$

Metadata

- Metadata is saved in the main memory
- A hashtable is used to assign the position of the page on the HDD to the position of the page on the SSD, if the page is existing on the SSD
- The temperature statistics are also saved with a region based hashtable
- To identify the coldest page a heap is maintained:
  - When the coldest page needs to be identified, the temperature of the root of the heap gets updated 5 times
  - After that, the root is chosen as the coldest page
  - The root may not be the absolute coldest page after the 5 updates, but the time needed is still logarithmic ($O(\text{n})$ otherwise with heapify()):

$$O(5 * \log(\text{n})) = O(\log(\text{n}))$$

Metadata

- Metadata is saved in the main memory
- A hashtable is used to assign the position of the page on the HDD to the position of the page on the SSD, if the page is existing on the SSD
- The temperature statistics are also saved with a region based hashtable
- To identify the coldest page a heap is maintained:
  - When the coldest page needs to be identified, the temperature of the root of the heap gets updated 5 times
  - After that, the root is chosen as the coldest page
  - The root may not be the absolute coldest page after the 5 updates, but the time needed is still logarithmic ($O(\mathrm{n})$ otherwise with heapify()):

$$O(5 * \log(\mathrm{n})) = O(\log(\mathrm{n}))$$

Metadata

- Metadata is saved in the main memory
- A hashtable is used to assign the position of the page on the HDD to the position of the page on the SSD, if the page is existing on the SSD
- The temperature statistics are also saved with a region based hashtable
- To identify the coldest page a heap is maintained:
  - When the coldest page needs to be identified, the temperature of the root of the heap gets updated 5 times
  - After that, the root is chosen as the coldest page
  - The root may not be the absolute coldest page after the 5 updates, but the time needed is still logarithmic ($O(n)$ otherwise with heapify()):

$$O(5 * \log(n)) = O(\log(n))$$

Replacement logic

The temperature based replacement algorithm was compared to:

- First in first out (FIFO)
- Least recently used (LRU)
- Clock
- Adaptive replacement (ARC)
- Optimal replacement (OPT / MIN)
- No cache at all

Replacement logic

The temperature based replacement algorithm was compared to:

- First in first out (FIFO):
  First page written is the first page to be replaced
- Least recently used (LRU)
- Clock
- Adaptive replacement (ARC)
- Optimal replacement (OPT / MIN)
- No cache at all

Replacement logic

The temperature based replacement algorithm was compared to:

- First in first out (FIFO)
- Least recently used (LRU):
  The page which was not used for the longest time gets replaced
- Clock
- Adaptive replacement (ARC)
- Optimal replacement (OPT / MIN)
- No cache at all

Replacement logic

The temperature based replacement algorithm was compared to:

- First in first out (FIFO)
- Least recently used (LRU)
- Clock:
  Every cached page has a flag indicating whether the page was accessed. A pointer goes through every page then and chooses the first page with disabled flag, but for each page pointed at with enabled flag the flag also gets disabled.
- Adaptive replacement (ARC)
- Optimal replacement (OPT / MIN)
- No cache at all

Replacement logic

The temperature based replacement algorithm was compared to:

- First in first out (FIFO)
- Least recently used (LRU)
- Clock
- Adaptive replacement (ARC):
  Two sets of pages: One for pages accessed recently and one for pages accessed often (page was in the first set and got accessed again). These two sets get resized automatically to better utilize the available space.
- Optimal replacement (OPT / MIN)
- No cache at all

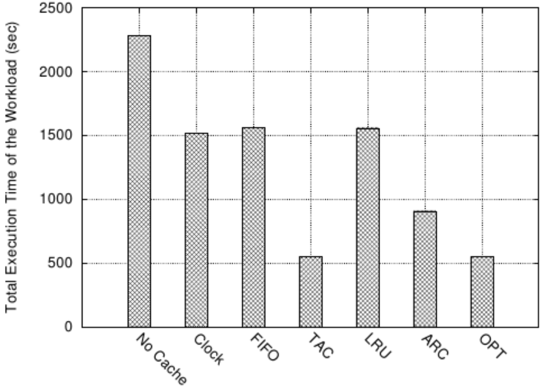
Replacement logic

The temperature based replacement algorithm was compared to:

- First in first out (FIFO)
- Least recently used (LRU)
- Clock
- Adaptive replacement (ARC)
- Optimal replacement (OPT / MIN):
  A offline-algorithm which calculates the best page replacement strategy by looking at all future page accesses.
- No cache at all

Replacement logic

Execution time with different page replacement algorithms



Background:

- Main memory bufferpool size: 160 MB
- SSD bufferpool size: 320 MB
- Database size: 15 GB
- Recorded TPC-H[1] query workload (500 queries)

1: TPC-H: Long running queries with high complexity

⇒ **TAC is the best online-algorithm in terms of execution time which means it is able to utilize the cache the best way**

System

Results for the whole system

- The execution of the testing transactions needed **up to 12x less execution time** with 8 queries running parallel (3x less with one query)
    - The speed-up is greater if there are more queries running parallel because the temperature informations are gathered faster and the SSD can be exploited further
- The SSD-Cache was capable of serving up to 83% of the reads without the HDD in this environment

Background:

- Main memory bufferpool size: 200 MB
- SSD bufferpool size: 1.2 GB
- Database size: 5 GB
- Working set size: 1.45 GB

System

Results for the whole system

- The execution of the testing transactions needed **up to 12x less execution time** with 8 queries running parallel (3x less with one query)
    - The speed-up is greater if there are more queries running parallel because the temperature informations are gathered faster and the SSD can be exploited further
- The SSD-Cache was capable of serving up to 83% of the reads without the HDD in this environment

Background:

- Main memory bufferpool size: 200 MB
- SSD bufferpool size: 1.2 GB
- Database size: 5 GB
- Working set size: 1.45 GB

Finding a cold page in constant time

Approach

- Finding the coldest page needs $O(\log(n))$ time
- Most of the time, only a sufficient cold page is needed, which is possible in $O(1)$

- Temperatur range gets divided into a fixed amount of bands realized as linked lists (50-100 bands are proposed)
- Pages get moved between linked lists when their temperature gets updated
- To find a cold page, the first non-empty band is searched from which the first element gets removed

Finding a cold page in constant time

Approach

- Finding the coldest page needs $O(\log(n))$ time
- Most of the time, only a sufficient cold page is needed, which is possible in $O(1)$

- Temperatur range gets divided into a fixed amount of bands realized as linked lists (50-100 bands are proposed)
- Pages get moved between linked lists when their temperature gets updated
- To find a cold page, the first non-empty band is searched from which the first element gets removed

Finding a cold page in constant time

Approach

- Finding the coldest page needs $O(\log(n))$ time
- Most of the time, only a sufficient cold page is needed, which is possible in $O(1)$

- Temperatur range gets divided into a fixed amount of bands realized as linked lists (50-100 bands are proposed)
- Pages get moved between linked lists when their temperature gets updated
- To find a cold page, the first non-empty band is searched from which the first element gets removed

Finding a cold page in constant time

Approach

- Finding the coldest page needs $O(\log(n))$ time
- Most of the time, only a sufficient cold page is needed, which is possible in $O(1)$

- Temperatur range gets divided into a fixed amount of bands realized as linked lists (50-100 bands are proposed)
- Pages get moved between linked lists when their temperature gets updated
- To find a cold page, the first non-empty band is searched from which the first element gets removed

Finding a cold page in constant time

- Besides the gain in efficiency, the approach leads to a smoothing-effect:
  - Pages that belong together are more likely to have the same temperature
  - Without the smoothing, already read pages belonging to a table scan may have a higher temperature than pages to be read. This leads to a higher probability of the upcoming pages to be overwritten.
- The downside of this extension is that it is only better in theory. Measurements led to the realization, that the heap-solution is better in practise.
- To combine the smoothing effect with the heap-solution, a new page only gets accepted if it is at least 1% warmer than the coldest page.

Finding a cold page in constant time

- Besides the gain in efficiency, the approach leads to a smoothing-effect:
  - Pages that belong together are more likely to have the same temperature
  - Without the smoothing, already read pages belonging to a table scan may have a higher temperature than pages to be read. This leads to a higher probability of the upcoming pages to be overwritten.

- The downside of this extension is that it is only better in theory. Measurements led to the realization, that the heap-solution is better in practise.
- To combine the smoothing effect with the heap-solution, a new page only gets accepted if it is at least 1% warmer than the coldest page.

Finding a cold page in constant time

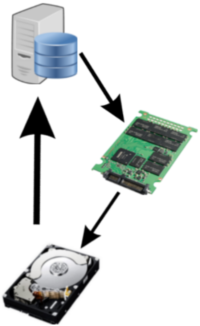
- Besides the gain in efficiency, the approach leads to a smoothing-effect:
  - Pages that belong together are more likely to have the same temperature
  - Without the smoothing, already read pages belonging to a table scan may have a higher temperature than pages to be read. This leads to a higher probability of the upcoming pages to be overwritten.

- The downside of this extension is that it is only better in theory. Measurements led to the realization, that the heap-solution is better in practise.
- To combine the smoothing effect with the heap-solution, a new page only gets accepted if it is at least 1% warmer than the coldest page.

Write-back caching

- It is also possible to include the previously presented write-back concept into the SSD-cache approach

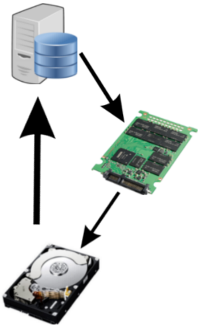
- 1st solution: A part of the SSD gets reserved for dirty pages. When it is filled, the pages get sorted by their location on the HDD and written to the HDD sequentially.



Write-back caching

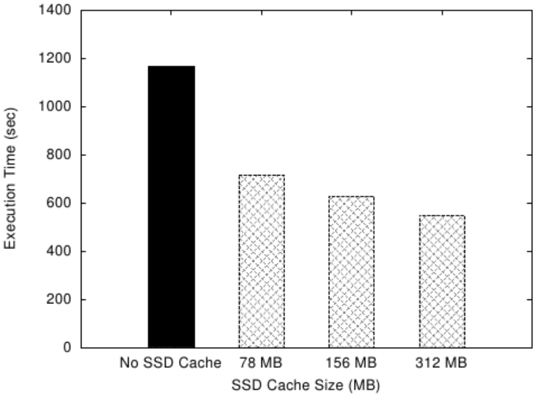
- It is also possible to include the previously presented write-back concept into the SSD-cache approach

- 2nd solution:
  - No space gets reserved and the pages are marked as *dirty* directly.
  - When a certain threshold is exceeded, the pages are sorted, written to the HDD and marked as *clean*.



Write-back caching

Results



⇒ **The speed-up of the system can be even greater by using write-back caching**

Background:

- Recorded TPC-C[1] query workload
- Only the writes are considered
- Writes are written to the SSD
- When the cache is filled, the pages are sorted and written to the HDD

1: TPC-C: Trading transactions

Outline

Motivation

The Problem

- Prior to the crash, the cache was filled with the data currently processed
- After the crash, the cache has to be handled as if it was empty because the informations about the content (metadata) were saved non-persistent in the main memory
- If the cache would still be available, it would likely contain the data needed during recovery

Motivation

The Problem

- Prior to the crash, the cache was filled with the data currently processed
- After the crash, the cache has to be handled as if it was empty because the informations about the content (metadata) were saved non-persistent in the main memory
- If the cache would still be available, it would likely contain the data needed during recovery

Motivation

The Problem

- Prior to the crash, the cache was filled with the data currently processed
- After the crash, the cache has to be handled as if it was empty because the informations about the content (metadata) were saved non-persistent in the main memory
- If the cache would still be available, it would likely contain the data needed during recovery

Motivation

Recovery

But how often does the system crash? Is the increased complexity worth it?

Situation:

- If the databasesystem crashes 3x a week ([3]) and each recovery-process takes 6 minutes (time without a cache for the system in the previous section)
- ... then the system is offline for around 16 hours per year only because of recovery
- This means for example that the employees can't work but still get paid on two days per year
  - A company with 100 employees relying on the database looses more than 50,000 Euro/year then

⇒ **The time needed for recovery is very important!**

Motivation

Recovery

But how often does the system crash? Is the increased complexity worth it?

Situation:

- If the databasesystem crashes 3x a week ([3]) and each recovery-process takes 6 minutes (time without a cache for the system in the previous section)
- ... then the system is offline for around 16 hours per year only because of recovery
- This means for example that the employees can't work but still get paid on two days per year
  - A company with 100 employees relying on the database looses more than 50,000 Euro/year then

⇒ **The time needed for recovery is very important!**

Motivation

Recovery

But how often does the system crash? Is the increased complexity worth it?

Situation:

- If the databasesystem crashes 3x a week ([3]) and each recovery-process takes 6 minutes (time without a cache for the system in the previous section)
- ... then the system is offline for around 16 hours per year only because of recovery
- This means for example that the employees can't work but still get paid on two days per year
  - A company with 100 employees relying on the database looses more than 50,000 Euro/year then

⇒ **The time needed for recovery is very important!**

Motivation

Recovery

But how often does the system crash? Is the increased complexity worth it?

Situation:

- If the databasesystem crashes 3x a week ([3]) and each recovery-process takes 6 minutes (time without a cache for the system in the previous section)
- ... then the system is offline for around 16 hours per year only because of recovery
- This means for example that the employees can't work but still get paid on two days per year
    - A company with 100 employees relying on the database looses more than 50,000 Euro/year then

⇒ **The time needed for recovery is very important!**

Motivation

Recovery

But how often does the system crash? Is the increased complexity worth it?

Situation:

- If the databasesystem crashes 3x a week ([3]) and each recovery-process takes 6 minutes (time without a cache for the system in the previous section)
- ... then the system is offline for around 16 hours per year only because of recovery
- This means for example that the employees can't work but still get paid on two days per year
  - A company with 100 employees relying on the database looses more than 50,000 Euro/year then

$\Rightarrow$ **The time needed for recovery is very important!**

Approaches

I'm going to describe three approaches which all aim at keeping the cache usable after the crash:

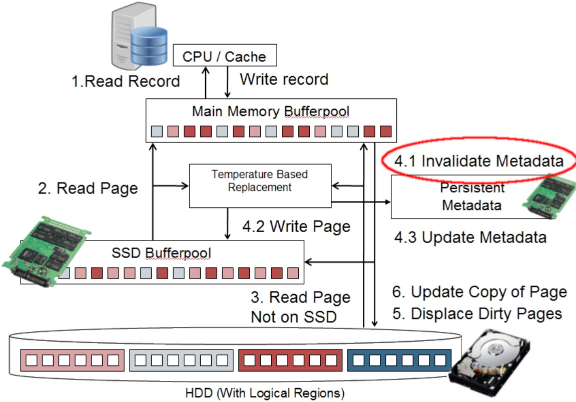
- Update-Write-Update (based on the SSD-Cache explained previously)
- Write-Update (used by Facebook)
- Lazy-Update Following an Update-Write

Update-Write-Update

System setup

Step 4.1: Invalidate Metadata

- New step!
- The first "Update"
- If a new page gets accepted by the replacement algorithm or a dirty page is updated the slot where it will be written in gets marked as *invalid*

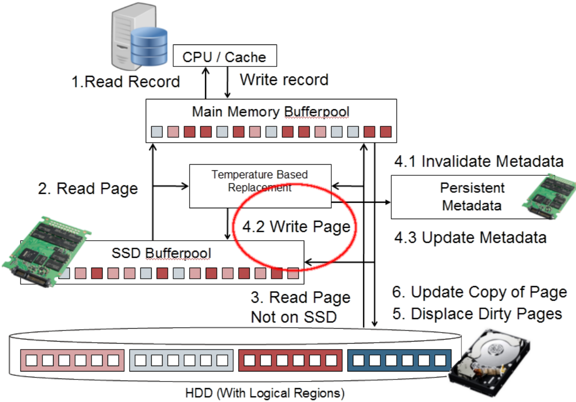


Update-Write-Update

System setup

Step 4.2: Write Page

- This step is not new
- The “Write”-part
- The new page is written to the SSD

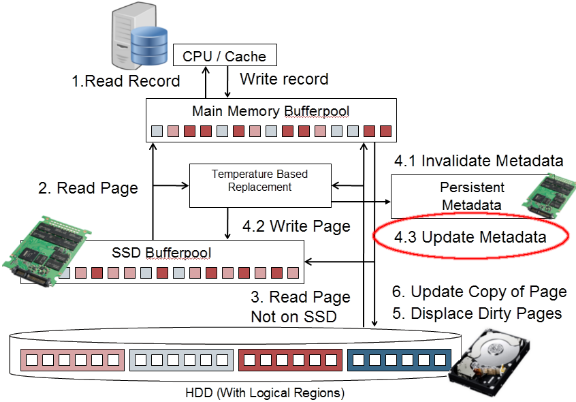


Update-Write-Update

System setup

Step 4.3: Update Metadata

- New step!
- The second "Update"
- After the write-process is finished, the written page gets marked as *valid*
- The whole process ensures that the pages on the SSD are always identical to those on the HDD



Update-Write-Update

Metadata

- The information about the SSD-slots and which pages they contain is saved to the SSD
  - Its size is less than 1% of the whole SSD-Cache, so it may even be saved on faster but also higher priced PCM-Memory
- The temperature statistics are also saved on the SSD so the cache can be used after a crash like it was used before the crash

Update-Write-Update

Metadata

- The information about the SSD-slots and which pages they contain is saved to the SSD
    - Its size is less than 1% of the whole SSD-Cache, so it may even be saved on faster but also higher priced PCM-Memory
- The temperature statistics are also saved on the SSD so the cache can be used after a crash like it was used before the crash

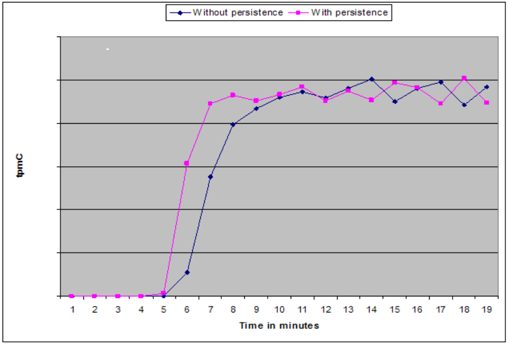
Update-Write-Update

Metadata

- The information about the SSD-slots and which pages they contain is saved to the SSD
    - Its size is less than 1% of the whole SSD-Cache, so it may even be saved on faster but also higher priced PCM-Memory
- The temperature statistics are also saved on the SSD so the cache can be used after a crash like it was used before the crash

Update-Write-Update

Results (1/4): Throughput after a crash



Background:

- Main memory bufferpool size: 1.2 GB
- SSD bufferpool size: 3.6 GB
- Database size: 48 GB
- TPC-C Benchmark

Results:

Without persistence:

- Recovery finishes after 5 minutes
- Performance stabilizes after 10 minutes

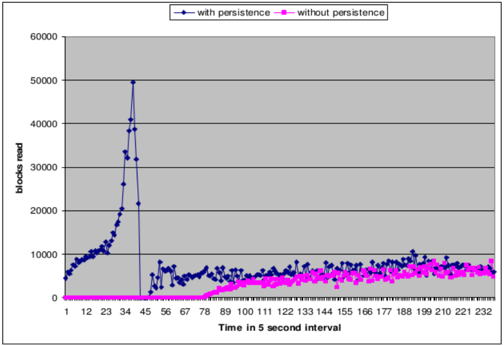
With persistence:

- Recovery finishes after 4 minutes
- Performance stabilizes after 8 minutes

$\Rightarrow$ **Recovery needs 20% less time with persistence**

Update-Write-Update

Results (2/4): Reads after a crash



Results:

Without persistence:

- SSD-Cache gets used after 390 seconds

With persistence:

- SSD-Cache gets used right from the beginning
- Most of the data needed gets read from the SSD
- The recovery-process finishes faster (compare the stable state)

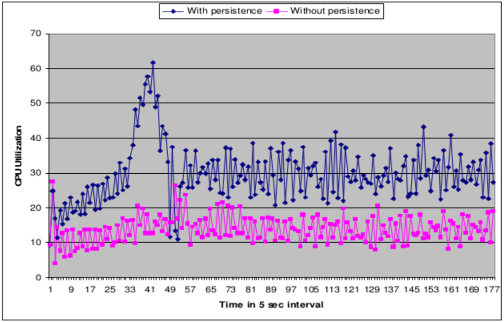
⇒ **The SSD-Cache is able to support the recovery-process**

Background:

- Main memory bufferpool size: 1.2 GB
- SSD bufferpool size: 3.6 GB
- Database size: 48 GB
- TPC-C Benchmark

Update-Write-Update

Results (3/4): CPU-load



Results:

Without persistence:

- The CPU-load sways around 15%

With persistence:

- The CPU-load sways around 30%

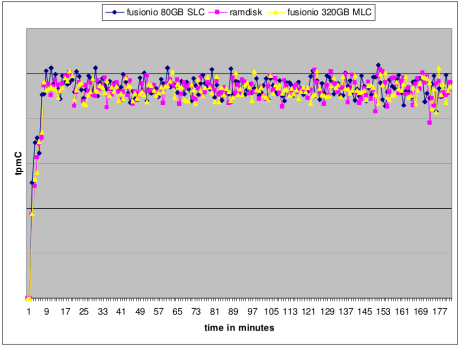
⇒ **The higher complexity also inceases the CPU-load**

Background:

- Dual core AMD CPU (3GHz)
- Main memory bufferpool size: 1.2 GB
- SSD bufferpool size: 3.6 GB
- Database size: 48 GB
- TPC-C Benchmark

Update-Write-Update

Results (4/4): Impact on the I/O performance



Result:

Without persistence:

- “ramdisk”

With persistence:

- “fusionio 80GB SLC”
- “fusionio 320GB MLC”

$\Rightarrow$ **In this scenario the I/O performance is the same with and without persistence**

Background:

- Main memory bufferpool size: 1.2 GB
- SSD bufferpool size: 3.6 GB
- Database size: 48 GB
- TPC-C Benchmark

Write-Update

The idea

- Write-Update is used by Facebook with their FlashCache-software
- Write-Update tries to drop the first “Update” (the invalidation) from the “Update-Write-Update”-approach
  - The purpose is to cut down the load on the system (IO- and CPU-load)
- Besides, the cache will also be used to buffer writes to the HDD
- In contrast to the previous approach, Write-Update does not use a temperature based algorithm (FIFO or LRU is used)

Write-Update

The idea

- Write-Update is used by Facebook with their FlashCache-software
- Write-Update tries to drop the first “Update” (the invalidation) from the “Update-Write-Update”-approach
  - The purpose is to cut down the load on the system (IO- and CPU-load)
- Besides, the cache will also be used to buffer writes to the HDD
- In contrast to the previous approach, Write-Update does not use a temperature based algorithm (FIFO or LRU is used)

Write-Update

The idea

- Write-Update is used by Facebook with their FlashCache-software
- Write-Update tries to drop the first "Update" (the invalidation) from the "Update-Write-Update"-approach
  - The purpose is to cut down the load on the system (IO- and CPU-load)
- Besides, the cache will also be used to buffer writes to the HDD
- In contrast to the previous approach, Write-Update does not use a temperature based algorithm (FIFO or LRU is used)

Write-Update

The idea

- Write-Update is used by Facebook with their FlashCache-software
- Write-Update tries to drop the first “Update” (the invalidation) from the “Update-Write-Update”-approach
    - The purpose is to cut down the load on the system (IO- and CPU-load)
- Besides, the cache will also be used to buffer writes to the HDD
- In contrast to the previous approach, Write-Update does not use a temperature based algorithm (FIFO or LRU is used)

Write-Update

The approach

Like in the first approach, only the differences to the SSD-Cache described earlier will be shown

A write to the SSD (dirty or updated page) is handled the following way:

1. Write the data to the SSD and mark it as *dirty* (the *dirty*-flag is not set before the write like with Update-Write-Update)
2. Write the dirty pages to the HDD with background-threads based on FIFO (or LRU) and in sequential order ...
    1. ... at an appropriate time
    2. ... or if a certain threshold is exceeded
    3. ... or if the data is buffered for too long (default: 15 minutes)
3. After the write of the dirty pages, mark them as *valid*

Write-Update

The approach

Like in the first approach, only the differences to the SSD-Cache described earlier will be shown

A write to the SSD (dirty or updated page) is handled the following way:

1. Write the data to the SSD and mark it as *dirty* (the *dirty*-flag is not set before the write like with Update-Write-Update)
2. Write the dirty pages to the HDD with background-threads based on FIFO (or LRU) and in sequential order ...
   1. ... at an appropriate time
   2. ... or if a certain threshold is exceeded
   3. ... or if the data is buffered for too long (default: 15 minutes)
3. After the write of the dirty pages, mark them as *valid*

Write-Update

The approach

Like in the first approach, only the differences to the SSD-Cache described earlier will be shown

A write to the SSD (dirty or updated page) is handled the following way:

1. Write the data to the SSD and mark it as *dirty* (the *dirty*-flag is not set before the write like with Update-Write-Update)
2. Write the dirty pages to the HDD with background-threads based on FIFO (or LRU) and in sequential order ...
    1. ... at an appropriate time
    2. ... or if a certain threshold is exceeded
    3. ... or if the data is buffered for too long (default: 15 minutes)
3. After the write of the dirty pages, mark them as *valid*

Write-Update

The approach

Like in the first approach, only the differences to the SSD-Cache described earlier will be shown

A write to the SSD (dirty or updated page) is handled the following way:

1. Write the data to the SSD and mark it as *dirty* (the *dirty*-flag is not set before the write like with Update-Write-Update)
2. Write the dirty pages to the HDD with background-threads based on FIFO (or LRU) and in sequential order ...
    1. ... at an appropriate time
    2. ... or if a certain threshold is exceeded
    3. ... or if the data is buffered for too long (default: 15 minutes)
3. After the write of the dirty pages, mark them as *valid*

Write-Update

Recovery

- Write-Update distinguishes between a forced reboot and a crash
- In case of a forced reboot, the metadata (and the *dirty*/*valid* flags alongside) gets flushed to the SSD and a flag indicating this is written
  - After the reboot both the *dirty* and the *valid* pages are used
- In case of a crash, no flag is written and only the *dirty* pages will be used
  - The *valid* pages can not be used because they may just got overwritten but the metadata (*dirty*-flag and page address on the HDD) was not saved yet
  - Partial writes will be detected by using checksums (address on the HDD needed)
  - Only around 14% of the cache can be used after a crash ([6])

Write-Update

Recovery

- Write-Update distinguishes between a forced reboot and a crash
- In case of a forced reboot, the metadata (and the *dirty*/*valid* flags alongside) gets flushed to the SSD and a flag indicating this is written
  - After the reboot both the *dirty* and the *valid* pages are used
- In case of a crash, no flag is written and only the *dirty* pages will be used
  - The *valid* pages can not be used because they may just got overwritten but the metadata (*dirty*-flag and page address on the HDD) was not saved yet
  - Partial writes will be detected by using checksums (address on the HDD needed)
  - Only around 14% of the cache can be used after a crash ([6])

Write-Update

Recovery

- Write-Update distinguishes between a forced reboot and a crash
- In case of a forced reboot, the metadata (and the *dirty*/*valid* flags alongside) gets flushed to the SSD and a flag indicating this is written
  - After the reboot both the *dirty* and the *valid* pages are used
- In case of a crash, no flag is written and only the *dirty* pages will be used
  - The *valid* pages can not be used because they may just got overwritten but the metadata (*dirty*-flag and page address on the HDD) was not saved yet
  - Partial writes will be detected by using checksums (address on the HDD needed)
  - Only around 14% of the cache can be used after a crash ([6])

Lazy-Update Following an Update-Write (“LUFUW”)

The idea

The idea of LUFUW is to combine the advantages of Update-Write-Update and Write-Update

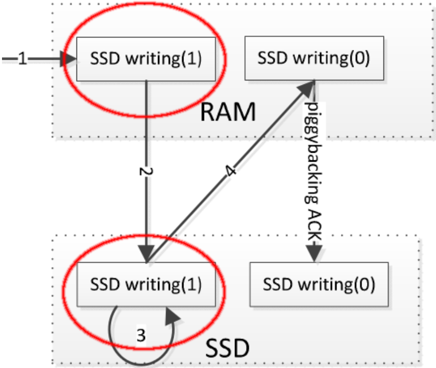
- With Update-Write-Update the whole cache can be used even after a crash
- Write-Update uses less operations and by this brings less load to the system

Lazy-Update Following an Update-Write ("LUFUW")

The approach

1st step: "Update"

- Write a *dirty*-flag indicating that the data is going to be written
- The flag is written to the copy of the metadata in the RAM and the SSD

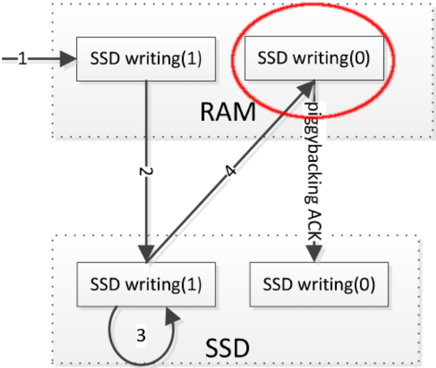


Lazy-Update Following an Update-Write ("LUFUW")

The approach

2nd step: "Write"

- After the page is written to the SSD, the *dirty*-flag is disabled but only inside the RAM
- After that, the write is reported as being finished to the system

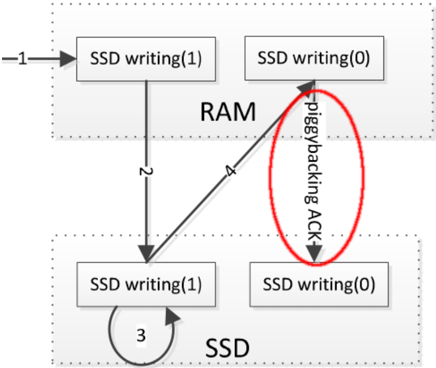


Lazy-Update Following an Update-Write ("LUFUW")

The approach

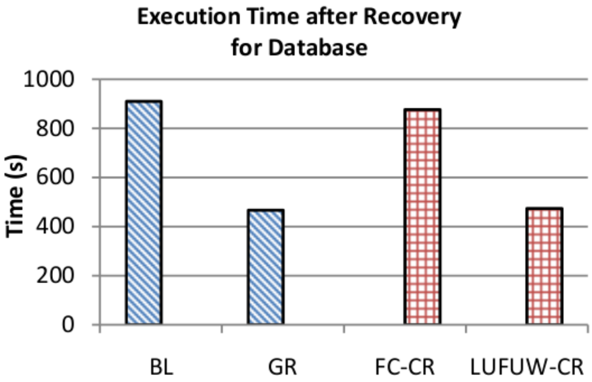
3rd step: "Lazy-Update"

- The next time step 1 is executed or other metadata is written to the same block on the SSD, the *dirty*-flag gets also reseted on the SSD
- This approach exploits the fact that one 4KB block of a SSD never only contains the metadata of one page (in this approach for example 240 metadata-entries fit in one block), but still with every metadata-update the whole block is overwritten



Lazy-Update Following an Update-Write (“LUFUW”)

Results



Background:

- BL: Restart with empty cache (LUFUW)
- GR: Restart with flushed metadata (LUFUW)
- FC-CR: Write-Update after a crash
- LUFUW-CR: LUFUW after a crash

- Database size: 100,000,000 rowsecsdr
- No information about RAM/Cache-size

⇒ **With LUFUW the recovery-process needs only around half of the time of Write-Update after a crash**

Outline

Summary and Bibliography

1. SSDs are fast and HDDs are cheap – therefore both should be combined!
2. SSD-Caching is a great way to make the databasesystem faster (up to 12x in some cases) through faster I/O
3. The needed time for recovery is critical, but with persistent SSD-Caching it can be reduced significantly

- BHATTACHARJEE, Bishwaranjan, et al. Enhancing recovery using an SSD buffer pool extension. In: Proceedings of the Seventh International Workshop on Data Management on New Hardware. ACM, 2011. S. 10-16.
- CANIM, Mustafa, et al. SSD bufferpool extensions for database systems. Proceedings of the VLDB Endowment, 2010, 3. Jg., Nr. 1-2, S. 1435-1446.
- HAERDER, Theo; REUTER, Andreas. Principles of transaction-oriented database recovery. ACM Computing Surveys (CSUR), 1983, 15. Jg., Nr. 4, S. 287-317.
- SRINIVASAN, Mohan; CALLAGHAN, Mark. FlashCache. 2010; http://github.com/facebook/flashcache
- SRINIVASAN, M.; SAAB, P. Flashcache: a general purpose writeback block cache for linux, 2011.
- YANG, Jing; YANG, Qing. A New Metadata Update Method for Fast Recovery of SSD Cache. In: Networking, Architecture and Storage (NAS), 2013 IEEE Eighth International Conference on. IEEE, 2013. S. 60-67.

Summary and Bibliography

1. SSDs are fast and HDDs are cheap – therefore both should be combined!
2. SSD-Caching is a great way to make the databasesystem faster (up to 12x in some cases) through faster I/O
3. The needed time for recovery is critical, but with persistent SSD-Caching it can be reduced significantly

- BHATTACHARJEE, Bishwaranjan, et al. Enhancing recovery using an SSD buffer pool extension. In: Proceedings of the Seventh International Workshop on Data Management on New Hardware. ACM, 2011. S. 10-16.
- CANIM, Mustafa, et al. SSD bufferpool extensions for database systems. Proceedings of the VLDB Endowment, 2010, 3. Jg., Nr. 1-2, S. 1435-1446.
- HAERDER, Theo; REUTER, Andreas. Principles of transaction-oriented database recovery. ACM Computing Surveys (CSUR), 1983, 15. Jg., Nr. 4, S. 287-317.
- SRINIVASAN, Mohan; CALLAGHAN, Mark. FlashCache. 2010; http://github.com/facebook/flashcache
- SRINIVASAN, M.; SAAB, P. Flashcache: a general purpose writeback block cache for linux, 2011.
- YANG, Jing; YANG, Qing. A New Metadata Update Method for Fast Recovery of SSD Cache. In: Networking, Architecture and Storage (NAS), 2013 IEEE Eighth International Conference on. IEEE, 2013. S. 60-67.

Summary and Bibliography

1. SSDs are fast and HDDs are cheap – therefore both should be combined!
2. SSD-Caching is a great way to make the databasesystem faster (up to 12x in some cases) through faster I/O
3. The needed time for recovery is critical, but with persistent SSD-Caching it can be reduced significantly

- BHATTACHARJEE, Bishwaranjan, et al. Enhancing recovery using an SSD buffer pool extension. In: Proceedings of the Seventh International Workshop on Data Management on New Hardware. ACM, 2011. S. 10-16.
- CANIM, Mustafa, et al. SSD bufferpool extensions for database systems. Proceedings of the VLDB Endowment, 2010, 3. Jg., Nr. 1-2, S. 1435-1446.
- HAERDER, Theo; REUTER, Andreas. Principles of transaction-oriented database recovery. ACM Computing Surveys (CSUR), 1983, 15. Jg., Nr. 4, S. 287-317.
- SRINIVASAN, Mohan; CALLAGHAN, Mark. FlashCache. 2010; http://github.com/facebook/flashcache
- SRINIVASAN, M.; SAAB, P. Flashcache: a general purpose writeback block cache for linux, 2011.
- YANG, Jing; YANG, Qing. A New Metadata Update Method for Fast Recovery of SSD Cache. In: Networking, Architecture and Storage (NAS), 2013 IEEE Eighth International Conference on. IEEE, 2013. S. 60-67.