

A New Cache Management Approach for Transaction Processing on Flash-based Database

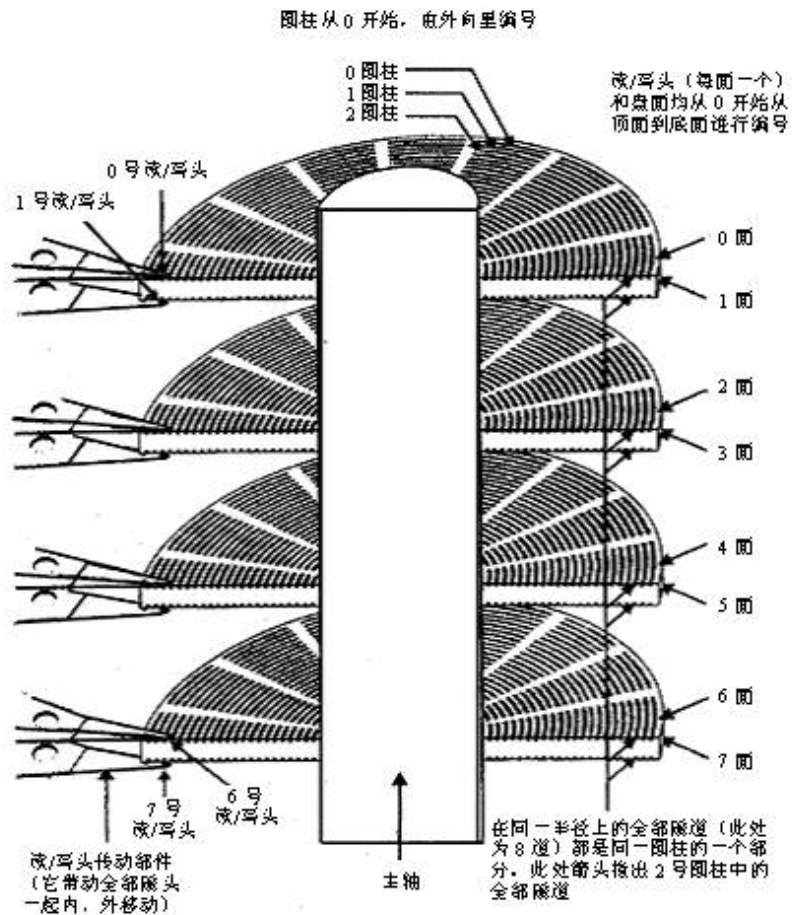
Da Zhou
2008-4-23



Outline

- Introduction of flash memory/magnetic disk
- Traditional IO mechanism on magnetic disk
- Problem statement
- Our method and Experiments
 - Committed transactions
 - Uncommitted transactions
 - Log-based write

Architecture of Hardware





Difference on Physical Characteristics

1. Read/Write/Erase
2. Asymmetric speed of read/write/erase
3. Erase before rewrite
4. Limited erase times
5. Read/write page, erase block, 1 block=64 pages

1. Read/Write
2. The same speed of read/write
3. Rewrite in-place
4. No limited write times
5. Read/write sector

Note: one flash page contains 2048 bytes



Outline

- Introduction to flash memory/magnetic disk
- Traditional IO mechanism on magnetic disk
 - Stealing frame
 - Not forcing pages
- Problem statement
- Our method and Experiments



Introduction to “Stealing Frame” Algorithm

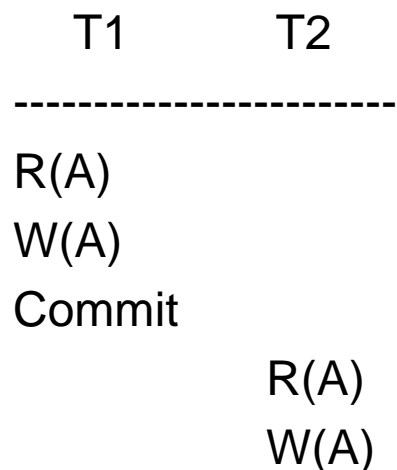
T1	T2

R(A)	
W(A)	
	R(B)

T1 reads A and updates A, then T2 read B. At this time, although T1 has not committed, the memory has no more capacity to load B. At this scenario, the updated A is written to disk, and B is loaded.



Introduction to “Not Forcing Pages” Algorithm



T1 reads A and updates A, then T1 commits. Although T1 has committed, memory still has large free space, so the updated A is still in memory. Another case: After T1 committed, then next transaction T2 need to read the result of the T1. At this case, the updated A will not be written to disk.



Outline

- Introduction to flash memory/magnetic disk
- Traditional IO mechanism on magnetic disk
- **Problem statement**
 - “Stealing frame” on flash memory
 - “Not forcing pages” on flash memory
 - LRU on flash memory
- Our method and Experiments

Basic Difference

The characteristics of the Server :

1. In-place update
2. High lateness on I/O
3. Large memory



The characteristics of the embedded computer:

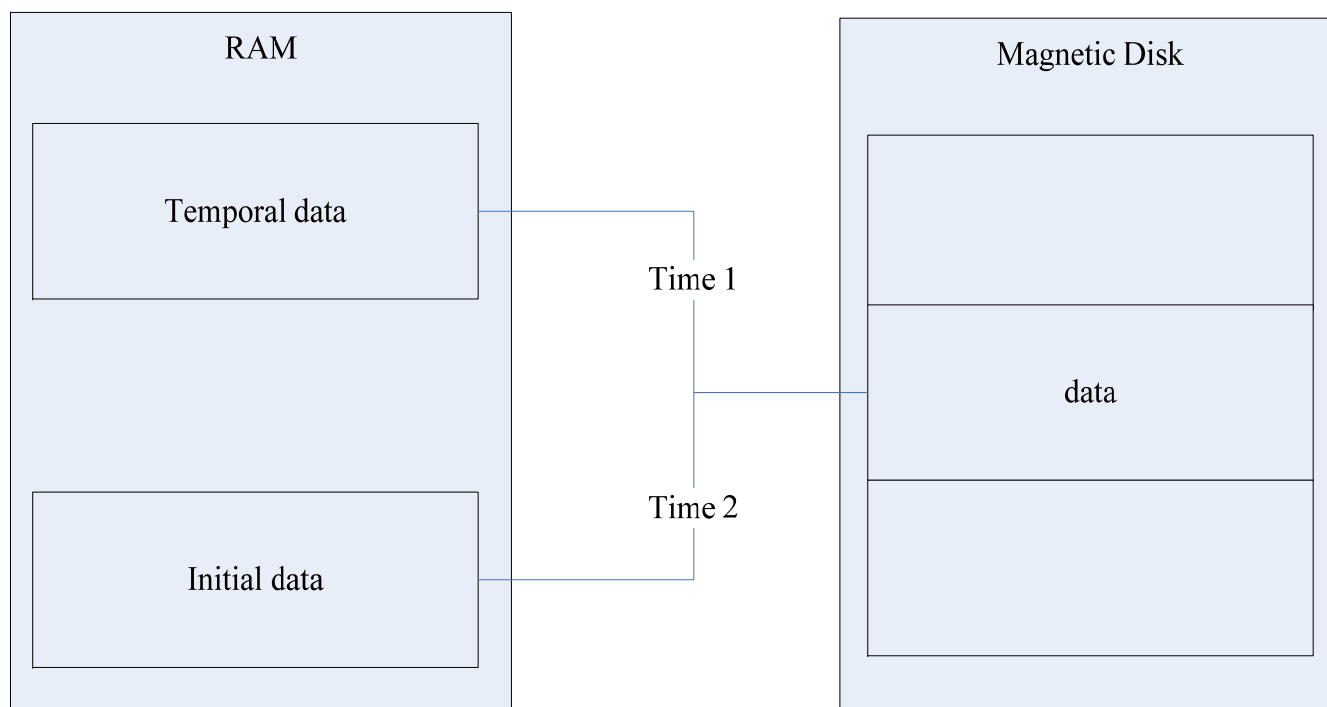
1. Out-of-place update
2. High read/write speed
3. Limited memory





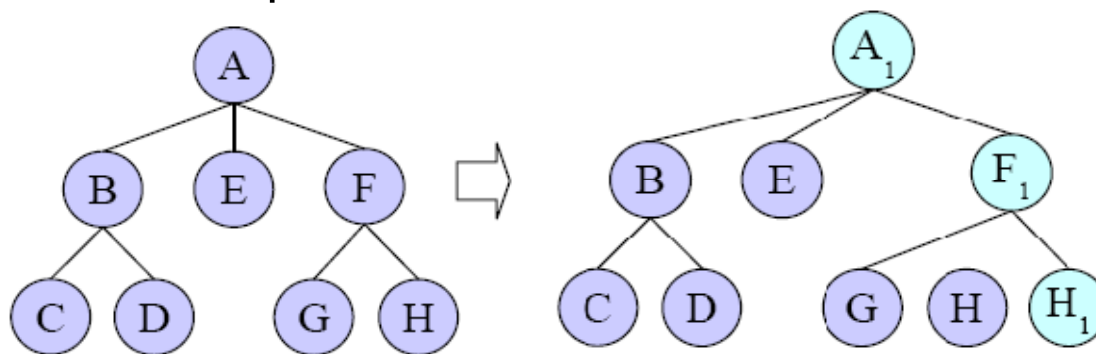
Stealing Frame (Magnetic Disk)

- IO Cost: 2 IOs



Stealing Frame (Embedded Computer)

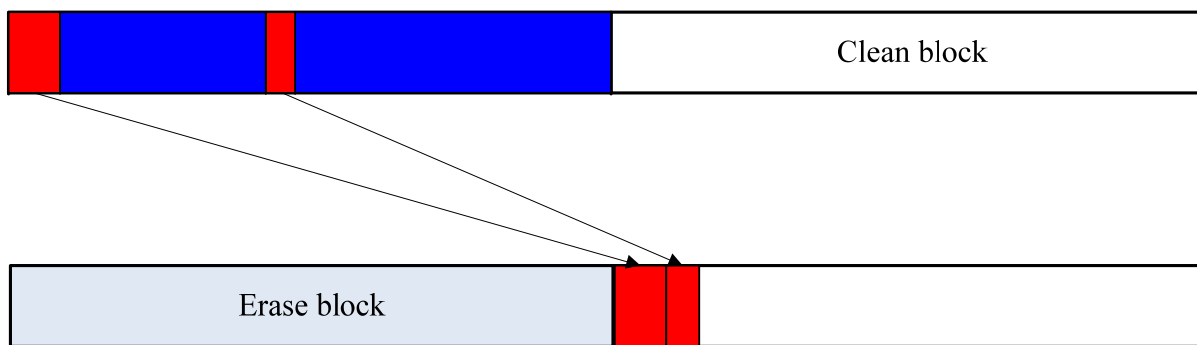
- Out-place-update
 - Series update



IO Cost

IO: depth

- Garbage collection



garbage:
depth*page

IO:depth write
depth/64 erase



Not Forcing Pages (Magnetic Disk)

- Large memory

Configuration options	HP 9000 Superdome 16 slots	HP 9000 Superdome 32 slots	HP 9000 Superdome 64 slots
Min./ Max. Memory	8 GB/512 GB	8 GB/1 TB	24 GB/2 TB

- High cost of IO

	160G 7200rpm
Average Seek Time	9ms

Not Forcing Pages (Flash Memory)

- Limited memory

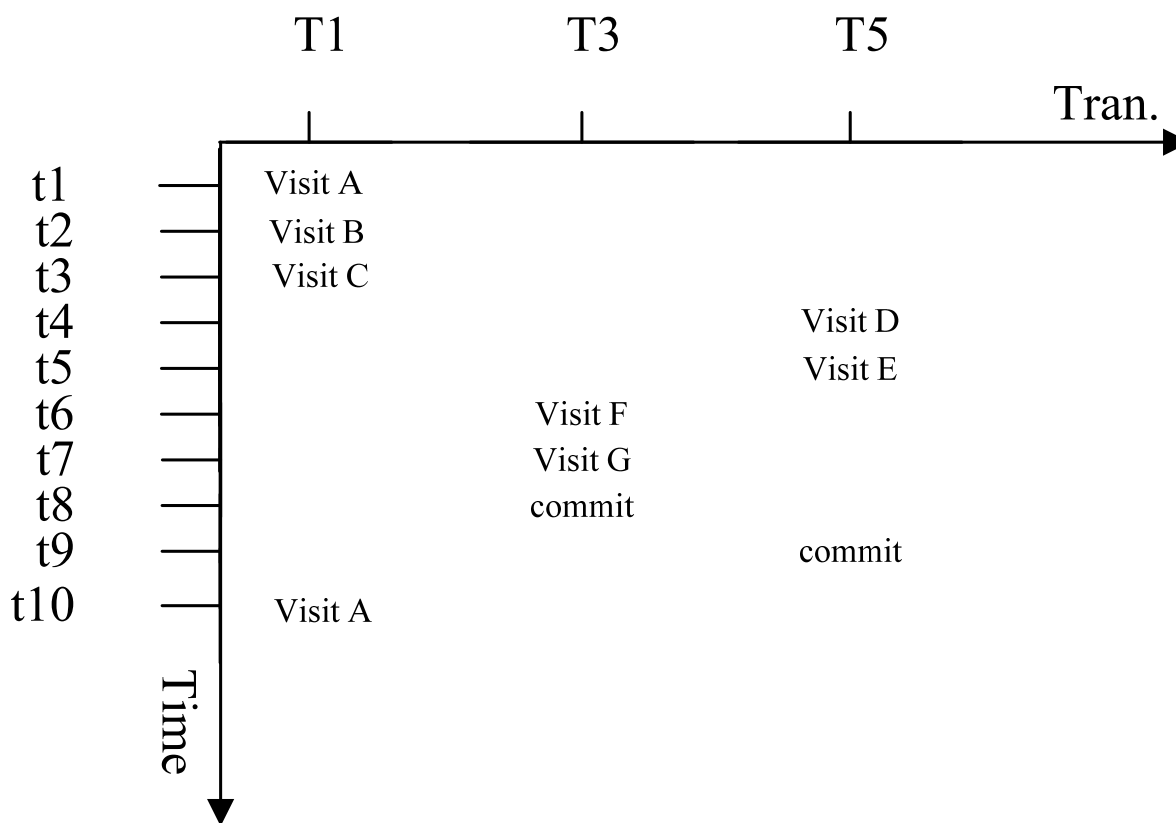
	HP iPAQ rw6818 Multimedia	HP iPAQ 112
Memory	64MB	64MB SDRAM

- Low cost of IO

	Flash memory	Magnetic disk
Read	25 μ s	9ms (Rand.)
Write	200 μ s	9ms (Rand.)
Erase	1.5ms	N/A



LRU on Disk



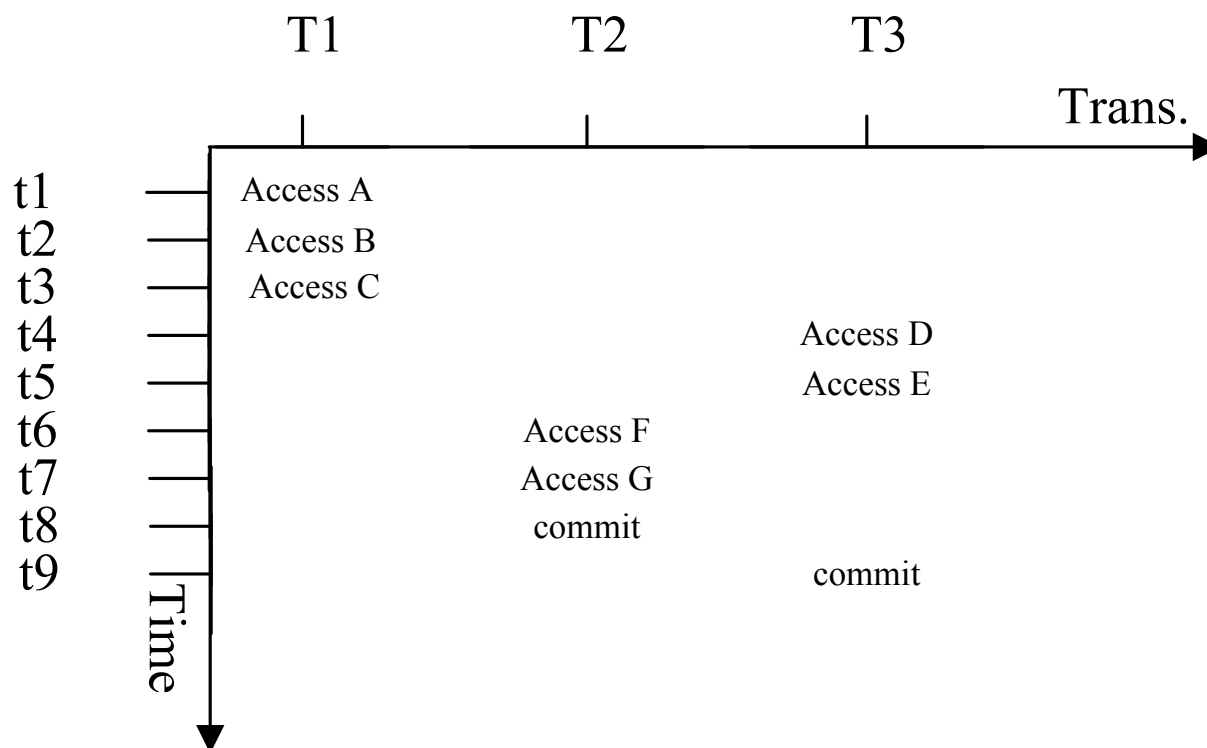
According to LRU, A will be firstly outputted to disk. If A is the next data to be read, series update and garbage collection will be unavoidable.



Outline

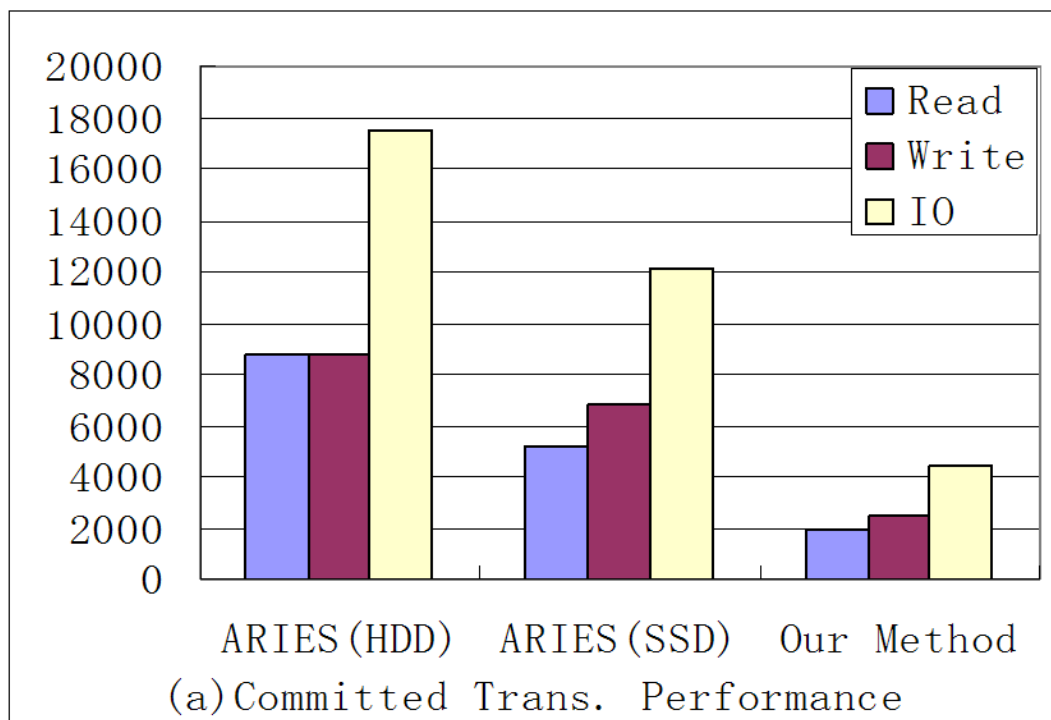
- Introduction to flash memory/magnetic disk
- Traditional IO mechanism on magnetic disk
- Problem statement
- **Our method and Experiments**
 - Committed Transactions
 - Uncommitted Transactions
 - Log-based write

Committed Transactions



Although T2 accesses F later than T3 accesses D, T2 will be outputted from memory earlier than T3 because T2 commits earlier than T3. According to our design, T3 is outputted earlier than T1.

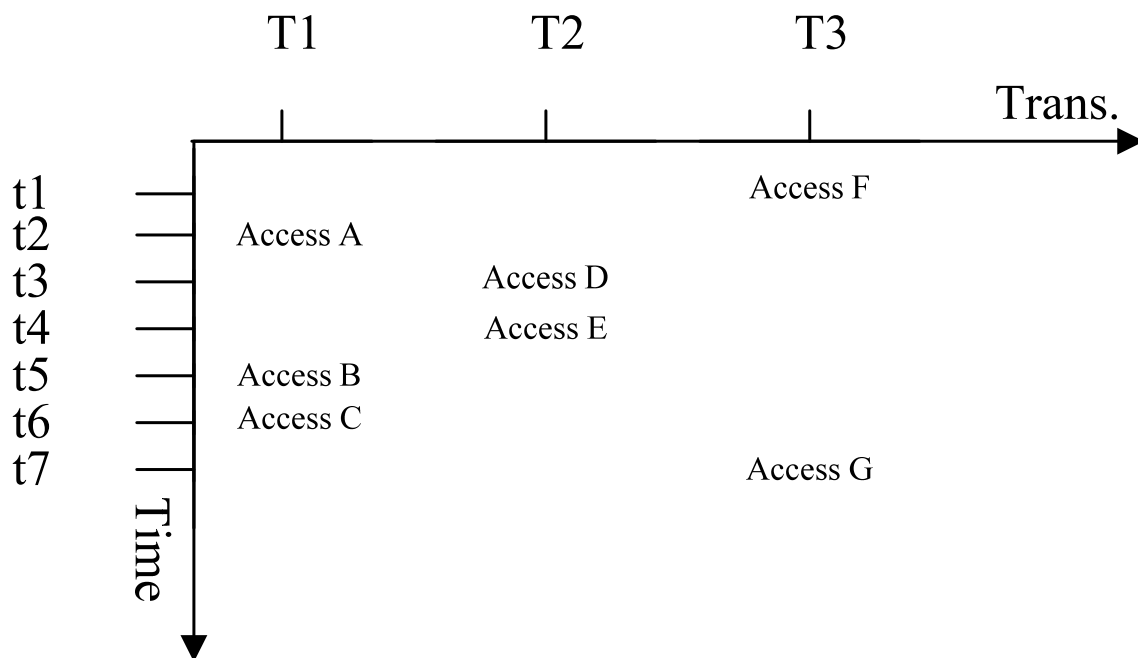
Committed Transactions



768 transactions are running in the RAM. When 256 transactions are committed, each of other 512 uncommitted transactions only loads half of the data.

According to the result, our method reduces by 63% IO time compared with the ARIES on SSD, even more on the write time.

Uncommitted Transactions

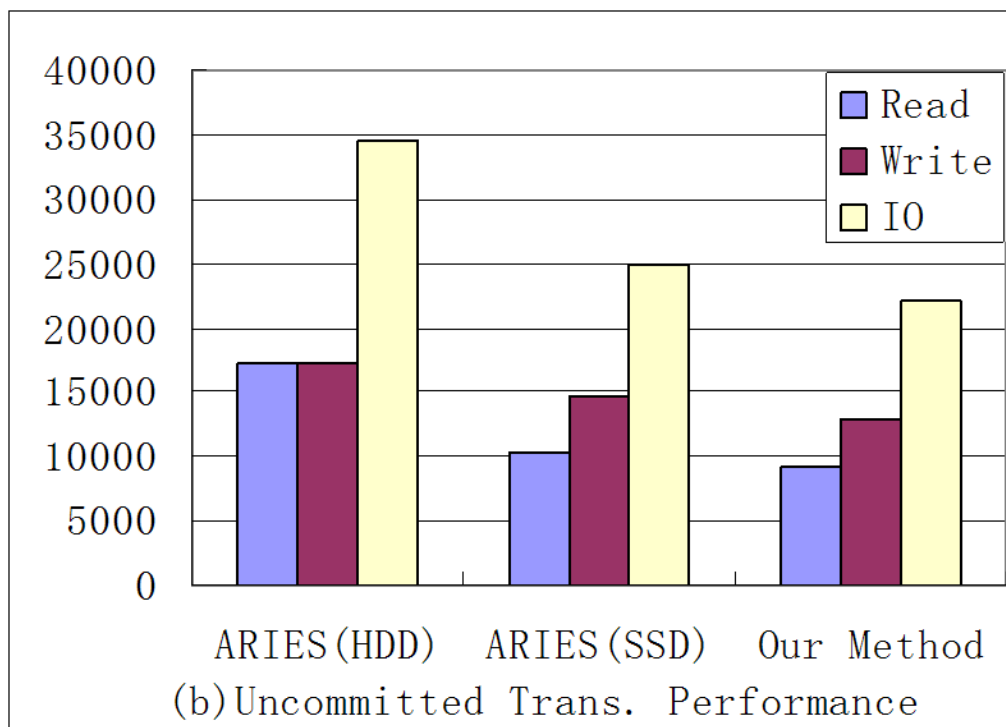


Average Access Time

$$AAT = \sum_{i=1}^n Ai / n$$

The AAT of T1, T2, T3 is 6.5, 3.5, 4, then the output order is T2, T3, T1. So our method reflects the real situation of the transaction replacement mechanism of the RAM.

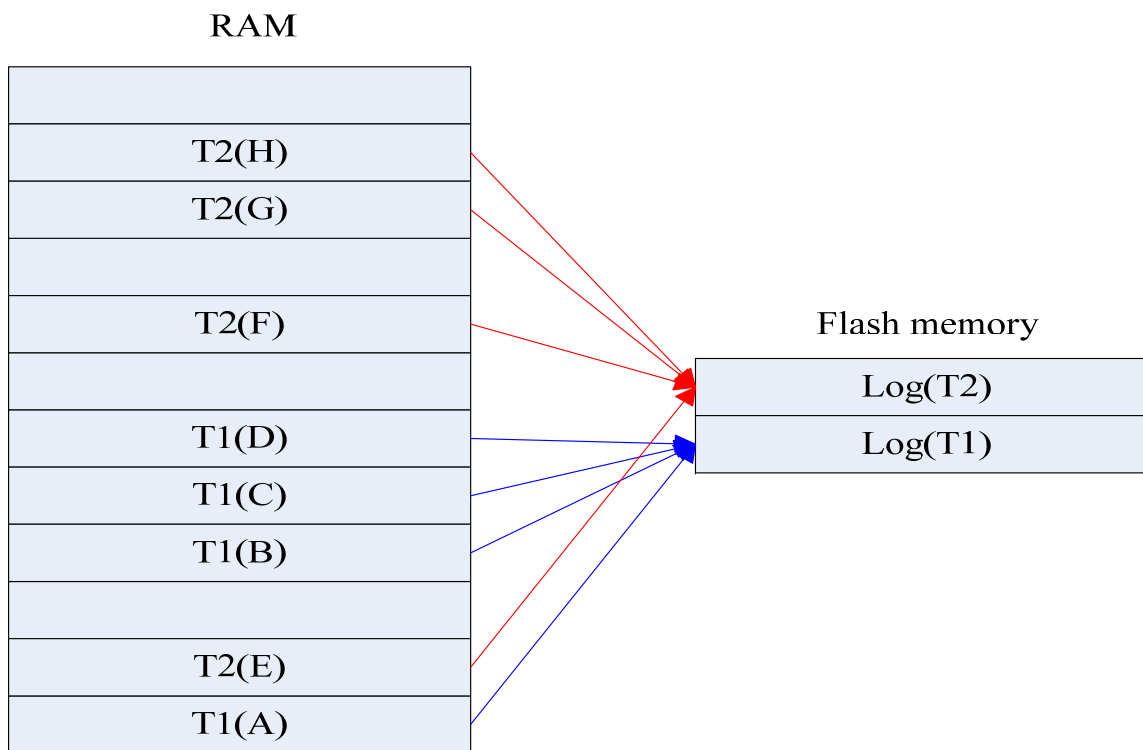
Uncommitted Transactions



1024 transactions run in the RAM at the same time. Every transaction loads a page of data in turn.

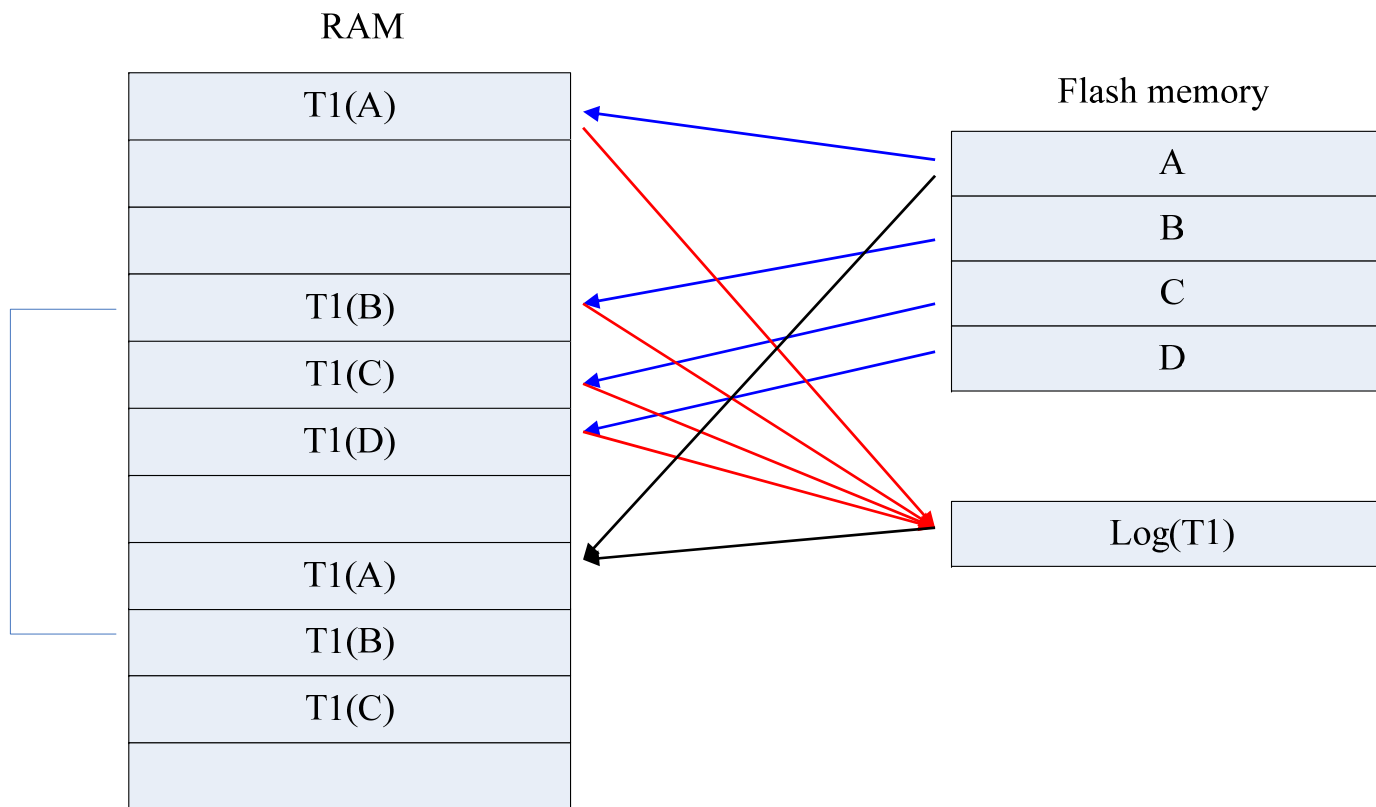
Our AAT-based method can enhance the replace right ratio, reduce the temporary data, and then improve by 11%

Log-based Write



When the transaction is outputted from the RAM, the modifications of transaction are organized as logs. The logs are written to the flash memory instead of data.

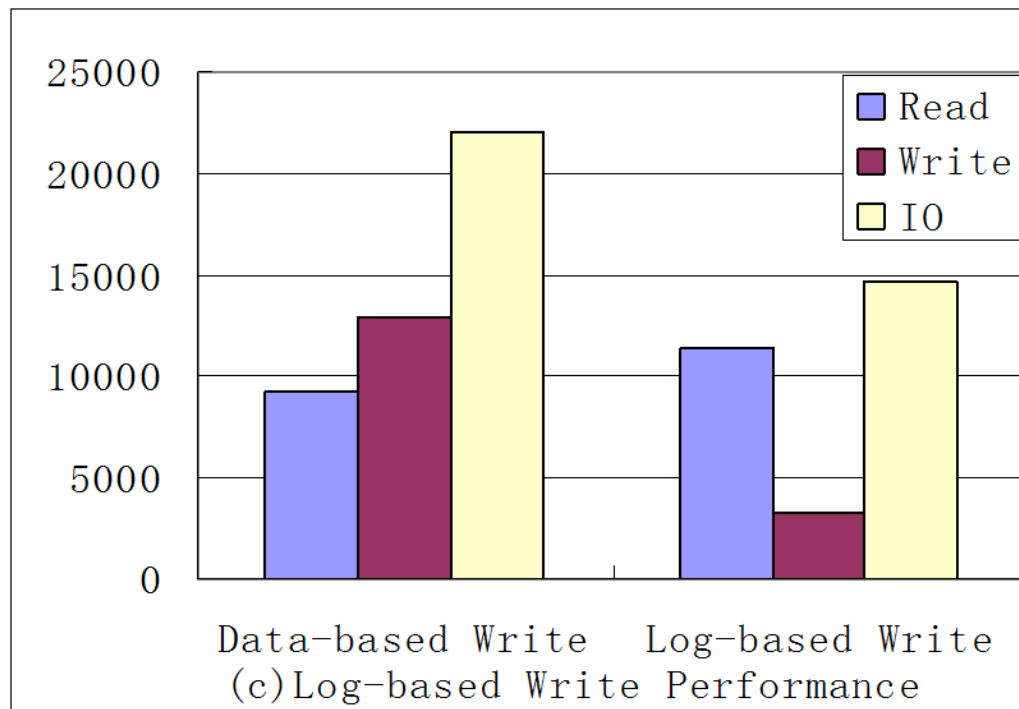
Log-based Write



If the transaction needs to read the data again, we must to read the log and the original data, merge the data and its log to get the newest data.

When the transaction is outputted from RAM, the logs are written to flash memory, and the data is not deleted in RAM. Only when new data is loaded into RAM, the data will be deleted.

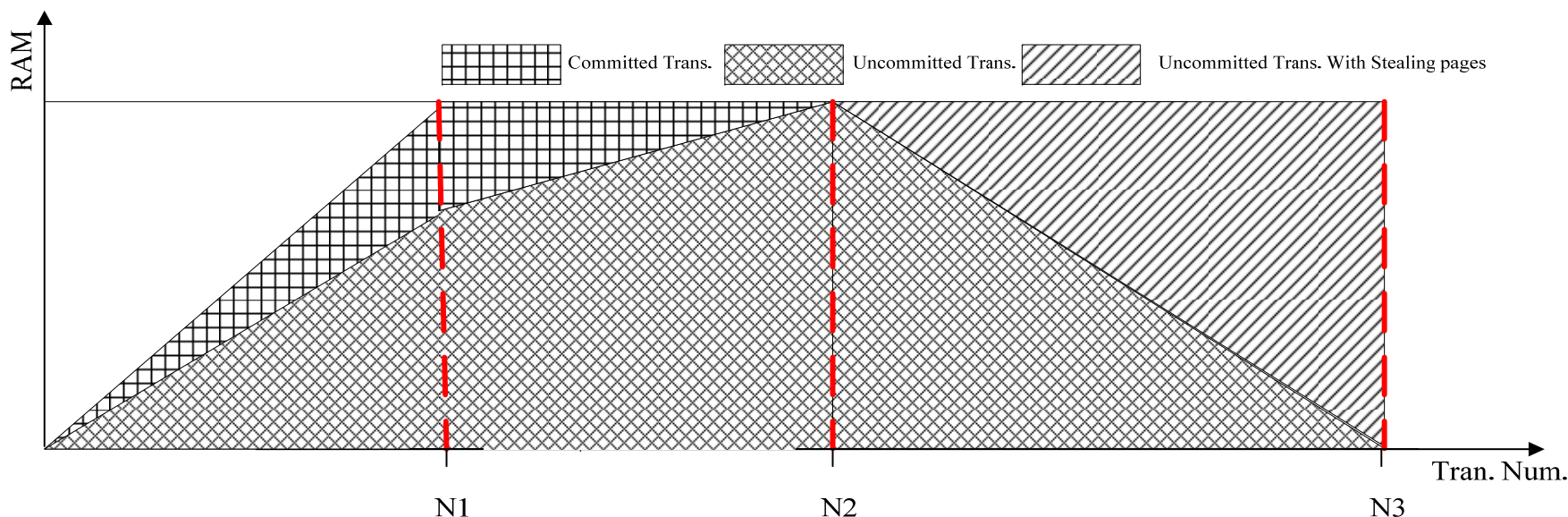
Log-based Write



1024 transactions run in the RAM at the same time. Every transaction loads a page of data in turn.

Although the performance of read increases by 25%, the total performance enhances by 34% as write performance improves by 75%.

Total Design



The uncommitted transactions have higher priority than the committed transactions in memory when transaction must be outputted to flash memory. As shown in figure 3, the number of committed transactions reduces and that of uncommitted transactions with stealing pages increases when the number of total transactions increases. When only uncommitted transactions are in memory, AAT is used to decide which transaction will be outputted to flash memory. When an uncommitted transaction is outputted, the modifications are formed as logs which are written to flash memory instead of the data.

Thank You 😊