A Probabilistic Data Replacement Strategy for Flash-Based Hybrid Storage System

Yanfei Lv, Xuexuan Chen, Guangyu Sun, and Bin Cui

School of Electronics Engineering and Computer Science, Peking University
Key Lab of High Confidence Software Technologies (Ministry of Education), Peking University
{lvyf,bin.cui,gsun,xuexuan}@pku.edu.cn

Abstract. Currently, the popularization of flash memory is still limited by its high price and low capacity. Thus, the magnetic disk and flash memory will coexist over a long period of time. How to design an effective flash-hard disk hybrid storage system emerges as a critical issue. Most of the existing works are designed based on traditional cache management approaches by taking the characteristics of flash into consideration. In this paper, we revisit the existing hybrid storage approaches and propose a novel probabilistic data replacement strategy for flash-based hybrid storage system, named HyPro. Different from traditional deterministic approaches, our approach moves the data probabilistically based on the data access pattern. Such a method can statistically achieve a good performance over massive memory operations of modern workloads. We also present the detailed data replacement algorithm and discuss how to determine the probability of data migration in the storage hierarchy consisting of main memory, flash, and hard disk. Extensive experimental results on various hybrid storage systems show that our method can yield better performance and achieve up to 50% improvements against the competitors.

1 Introduction

Although most of the people believe that the magnetic disk will be replaced by flash-based solid state drives (SSD) in the future, currently the popularization of flash memory is still limited by its high price and low capacity. Hence magnetic disk and flash memory will coexist over quite a long period of time. From the comparison in Table 1, flash displays a moderate I/O performance and price per GB between DRAM and hard disk. Consequently, it is straightforward to adopt flash memory as a level of memory between the HDD and main memory because of its advantages in performance [18,19]. The flash-hard disk hybrid storage is more and more adopted. Seagate provides a mixed storage hard disk with 4GB flash chip to improve the overall performance [2]. Windows operating system support Quick Boost from vista to accelerate the booting [1]. In addition, some companies start to replace some of the hard disk to SSD to build a hybrid storage system. In this case, how to design an effective flash-hard disk hybrid storage system emerges as a critical issue.

Y. Ishikawa et al. (Eds.): APWeb 2013, LNCS 7808, pp. 360–371, 2013.

Table 1. Comparison on different storage media

	Price($)	Capacity(GB)	Price($)/GB	Read(μs)	Write(μs)
DRAM(DDR3)	23.99	4GB	6.00	1	1
SSD	114.00	64GB	1.78	271	2012
Disk(7.2K)	119.99	1TB	0.12	12700	13700

The data migration among main memory, flash and disk is the most important issue in hybrid storage design. Many works has been done on this problem. TAC [7](Temperature-Aware Caching) adopts temperature to determine page placement. In TAC, a global temperature table is maintained for each page. The temperature of a page is decided by its access numbers and patterns in a period. The pages with higher temperature are placed in main memory and flash memory while a cold page evicted from main memory will be replaced to disk.

In contrast, the LC [11](Lazy Cleaning) and FaCE [13](Flash as Cache Extension) always cache a page exit from main memory to flash memory. LC handles flash memory as a write-back cache. The dirty pages are kept on flash first and if the percentage of dirty pages exceeds a threshold, these pages will be flushed to disk. Whereas FaCE proposes FIFO replacement for flash memory management which is an ideal pattern for flash write. In this way, FaCE can improve the throughput and shortens the recovery time of database.

All the above methods are all in deterministic way. In some cases, such deterministic migration policy is really inefficient. For example, in some cases, pages are only accessed once, and thus it is suboptimal to keep these cold pages on the flash memory as designed in LC and FaCE. The temperature method in TAC works well on hot page detection with stable pattern. However, on a workload changing, TAC takes a rather long time to forget the history and learn the new pattern. Another disadvantage of TAC is its high time and space consumption.

In order to overcome the problems in prior approaches, we propose a probability-based policy named HyPro to manage data storage and migration in the storage hierarchy, which is composed of main memory, SSD and HDD. In our approach, the priority of data in each level of the hierarchy is maintained separately. The key difference from prior work is that the data migration among different levels are no longer deterministic but based on probabilities. Compared to prior deterministic approaches, HyPro has several advances:

- Better management efficiency. The probabilistic data migration can be considered as a statistical frequency-based implementation. Since existing cache policies can also be applied to SSD in our approach, HyPro is a seamless combination of the cache management and frequency-based migration policies, which can achieve better management efficiency so that the total I/O performance can be improved.
- Less unnecessary data replacement. For deterministic approaches, there may be some unnecessary data movements for cold pages. In our stochastic based

scheme, such unnecessary data replacement can be effectively eliminated with the control of probabilities.
- Lower overhead. To achieve the hot page detection, prior work needs to maintain a global list to identify the hottest and coldest pages. The space complexity is $O(n)$ and the time complexity of the maintainable no less than $O(log(n))$, where n is the number of all pages accessed including those on the disk. In our approach, the space complexity is negligible and the time complexity is $O(1)$.

The remainder of the paper is organized as follows. Related work is described in Section 2. Section 3 describes our framework. Parameter tuning method is presented in Section 4. Experimental results are shown in Section 5, and we make a conclusion in Section 6.

2 Related Work

Nowadays, flash-based hybrid storage has been gradually recognized as an economical way for a practical system by more and more people. Some hard disks leverage a small flash memory to improve I/O performance [5]. With the increment on the capacity, SSD is more and more widely deployed in storage systems. The early SSD is skilled in reading but uncompetitive in writing. Thus migration methods [19,14] are proposed to dynamically transfer read intensive pages to flash and write intensive ones to disk. Recently, SSD thoroughly surpasses disk on both read and write speed, and hence the popular method is to adopt flash as a middle-level cache between disk and main memory. Existing work includes static deployment and dynamic loading. An object placement method [6] is developed to give a proper deployment for the components of DBMS. By comparing the object performance on SSD and disk beforehand, those with higher benefit per size are chosen to place on SSD. Other methods suggest putting certain part of the system to flash. FlashLogging [8] illustrates that storing the log of DMBS to flash can largely improve the overall performance. Debnath et al. propose [9,10] FlashStore and SkimpyStash to discuss the proper way to put the key-value pair to SSD. The static methods need to know the specific information about the application and is not self-adaptive to various environments. Some methods based on dynamically page transferring are proposed. TAC(Temperature-Aware Caching) [7] is the dynamic version of object placement strategy. It allocates temperature to the extents according to access pattern and I/O cost and keeps the data with higher temperature to higher level of the storage structure. Researchers from Microsoft [11] discuss several possible designs for hybrid storage methods. According to testing the LC(Lazy-Cleaning) is the best design. LC method shows better performance than TAC on write intensive traces and similar on read-intensive traces. hStorageDB [16] adopts semantic information to exploit the capability of hybrid storage system, which is from another aspect to solve the hybrid storage problem. FaCE [13] proposes to use the flash in FIFO manner to improve throughput and provide faster recovery.

3 Probabilistic Framework

3.1 The HyPro Approach

The typical structure of a hybrid storage system is illustrated in Figure 1. All the data is stored on hard disk and organized as data pages. A page need to be loaded into main memory before being accessed. Since flash has better performance compared to disk, it works as the level between main memory and disk. When a page miss happens in main memory, the flash will be checked first. The disk is only accessed when the page is not found in the flash.

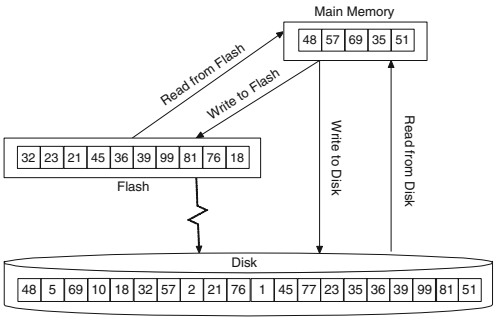


Fig. 1. Illustration of hybrid storage system

In this system, the data placement and migration is a critical issue to achieve better performance. In this part, we introduce our probabilistic approach for hybrid storage management, named HyPro. The overall structure of HyPro is shown in Fig 2. In our framework, the pages in main memory and flash memory are exclusive from each other. In other words, we do not keep a page in flash memory if it is already in main memory.

In the HyPro, we adopt two probabilities to control the data migration. If some of the pages on flash are frequently accessed, it's better to be elevated to main memory. We call this process *elevation*. Once the elevation happens we have to evict a page from main memory. Obviously, the elevation should be managed carefully so that the benefits of accessing hot pages can offset the I/O cost overhead caused by data movement. In our probabilistic data management, we use a probability named $p_{elevate}$ to control the elevation frequency. As shown in Fig 3, when a page is accessed, it has the chance of $p_{elevate}$ to be kept in main memory; otherwise, this page will be evicted on the next data access. It is obvious that the page has more chance to be elevated if it is more frequently used. Hence, real hot pages are detected and promoted into main memory statistically during a long runtime. In each elevation, a cold page in main memory need to be evicted to flash and placed in the original space of the elevated page.

In the HyPro, some page may be evicted from the main memory, which we name *sinking* operation in this paper. At first glance, this page is likely to be

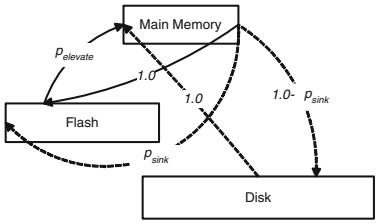
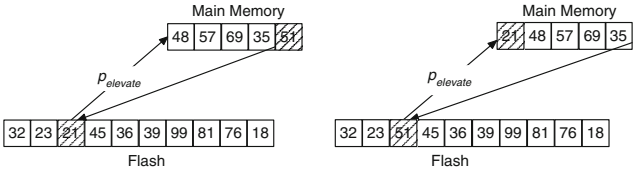


Fig. 2. Overall of Probabilistic Framework



(a) Before elevation (b) After elevation

Fig. 3. An elevation example

hotter than pages on flash memory, and should replace one flash page. However, sinking to flash incurs a flash write. Whether the future benefit is worth this write depends on the cost ratio between flash read and write as well as the hotness of the page being sunk. An example of sinking is illustrated in Fig 4. We take a probability p_{sink} to control the ratio of sinking to flash. A main memory evicted page has chance of p_{sink} to replace a flash page, otherwise, it will be discarded directly or written back to disk if it is dirty. Let's see why this works. We assume the main memory evicted page is M (page 51 in Fig 4 (a)), and the replaced page from flash is F(page 18 in Fig 4 (a)), respectively. The larger p_{sink} is, the more evicted pages are sunk to flash, and the closer the hotness of M and F are. Consequently, the benefit of sinking to flash will be small for close hotness of M and F. By setting proper value of $p_{elevate}$ and p_{sink}, we can achieve a better trade-off between main memory and flash accesses and the overall I/O cost diminishes. We will talk about the parameter tuning in the Section 4.

The pseudocode of HyPro are listed in Algorithm 1. A structure named frame is used to store the position of each page, and the frames are organized in a hash table to facilitate searching. The Algorithm 1 illustrates the routine of page access. First, the position of the page is determined, and then different operations are conducted according to the page position. The Algorithm 1(line 13) may invoke Algorithm 2. Algorithm 2 loads a page from disk and puts this page to the right position according to the p_{sink}. Although LRU is adopted in our experiments for main memory and flash memory management, HyPro can support other strategies such as LIRS [12] and ARC [17]. The HyPro is easy to

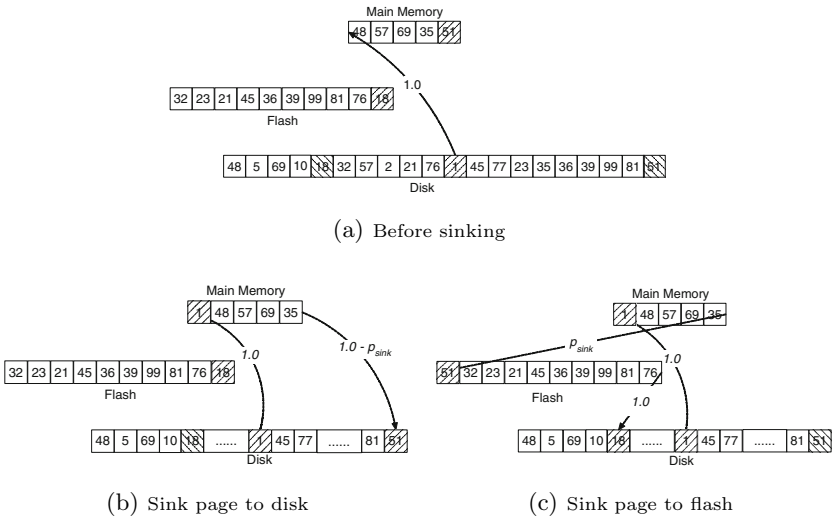


Fig. 4. A sink example

implement and quick enough for online processing. The time complexity is $O(1)$ for each page access.

In this paper, we focus on the data migration design. Nevertheless, some optimizations can be supported in HyPro applied to further improve the performance e.g. by considering the asymmetric I/O and the access pattern (random/sequential). For example, if the asymmetric I/O is considered, different probabilities can be allocated to read and write operations respectively, which can make one write operation equivalent to the effect of n read operations. Furthermore, we could also manage flash in FIFO manner as FaCE to transform flash space allocation into sequential pattern.

4 Parameter Tuning

The probabilities of transforming data among different memory levels are the crucial part in our stochastic page management policy. In this section, we provide study on how to automatically tune these probabilities based on the "cost" analysis. Table 2 facilitates fast check on the notations used in this section.

Definition 1. *For a cache management algorithm, we denote the place of the page to be evicted as the "evict position". (For example, the end of the LRU queue).* N_{evict} *is defined as the total hit number on the "evict position".*

To begin with, a definition is introduced. Note that we consider the hit number on a "position" instead of on a specific page in this definition. For example, if LRU algorithm is adopted in main memory, the N_{evict} stands for the number of accesses on the LRU end. If a read operation is hit on the least recently used

Algorithm 1. The HyPro algorithm

```
Input: an operation request on page p
if p exists in main memory then
    Perform read/write operation in main memory;
else if p exists in Flash memory then
    if rand() < p_elevate then
        AcquireFreeMemPage();
        write the evicted page to flash memory if any;
        read from Flash memory, or just write to main memory;
    else
        Perform read/write operation towards Flash memory;
    end
else
    call OnMemFull algorithm when main memory is full;
    AcquireFreeMemPage();
    read from disk, or just write to main memory;
end
```

Algorithm 2. The OnMemFull algorithm

```
Output: a free page in memory
if memory is full then
    get a victim page from the buffer manager;
    if rand() < p_sink then
        AcquireFreeFlashPage();
        flush the page to Flash memory;
    end
    flush the page to disk if dirty;
    return the page;
end
return a free page;
```

Table 2. Parameters used in this section

Parameters	Description
C_{dr},C_{fr}	The read time cost of one page on disk, flash respectively
C_{dw},C_{fw}	The write time cost of one page on disk, flash respectively
$p_{elevate}$	The probability to elevate
p_{sink}	The probability to sink
R_{fevict},R_{mevict}	The number of read hit on the evict position of flash, main memory respectively
W_{fevict},W_{mevict}	The number of write hit on the evict position of flash, main memory respectively

page P, N_{evict} increases. However, at this time page P is moved to LRU head and a new page named P' is moved to LRU end. Then next time we increase N_{evict} when the P' is accessed rather than P.

Assume that in a certain time period, a page P_f on flash is read R_f times and written W_f times. R_{mevict} and W_{mevict} stand for the read and write times on the "evict position" in the main memory during the same time period. The read and write costs of flash are denoted as C_{fr} and C_{fw}.

In HyPro, p_{sink} is adopted to balance the eviction to flash and disk. Hence, we discuss the tuning of p_{sink} by comparing the cost of the two cases: 1)evict the page to flash (Figure 4 (c)) and 2)evict the page to disk (Figure 4 (b)). The I/O costs of two cases are calculated as follows respectively.

Case 1. Evict page P_{mevict} to flash, and if flash is full evict the P_{fevict} to disk (when dirty). Consequently, P_{mevict} will be accessed from flash and P_{fevict} from disk, and thereby the corresponding I/O cost is:

$$C_{sinkf} = R_{mevict}C_{fr} + W_{mevict}C_{fw} + R_{fevict}C_{dr} + W_{fevict}C_{dw} + C_{fw} \quad (1)$$

Case 2. Evict page P_{mevict} to disk. In this case P_{mevict} will be accessed from disk while the P_{mevict} mentioned in Case 1 is still accessed from flash. Thus the I/O cost is:

$$C_{sinkd} = R_{mevict}C_{dr} + W_{mevict}C_{dw} + R_{fevict}C_{fr} + W_{fevict}C_{fw} \quad (2)$$

In the case of $C_{sinkf} < C_{sinkd}$, it is more I/O efficient to evict a page to flash, and hence, p_{sink} should be increased and vice versa. The above analysis only takes the I/O cost of page transferred, that is, page evicted and page elevated into consideration. Actually, the I/O costs of other pages will also be influenced which are not the primary cost and experiments show that the obtained C_{sinkf} and C_{sinkd} can deal with parameter adjusting effectively.

The C_{sinkf} and C_{sinkd} are very small and unstable for a single page on a short period of time. In practice, we accumulate C_{sinkf} and C_{sinkd} on all the evicted page in a certain window on the trace, so that the p_{sink} can be adjusted based on the comparison of C_{sinkf} and C_{sinkd} on each accumulation. Note that the calculation and parameter tuning described above need only $O(1)$ time for each access. The tuning will not significantly increase the overhead of whole strategy.

The tuning of $p_{elevate}$ can be performed in a similar way with p_{sink}, and thus is omitted here due to space limitation.

5 Performance Evaluation

In this section, we conduct a trace-driven simulation to evaluate the effectiveness of the proposed framework. The traces used include TPC-B, TATP [3] and making Linux kernel (MLK for short) to further evaluate the performance on various workloads. TAC and FaCE are chosen as the competitors. The simulation

is developed in Visual Studio 2010 using C#. All experiments are run on a Windows 7 PC with a 2.4 GHz Intel Quad CPU and 2 GB of physical memory.

5.1 Experimental Setup

We use three traces mentioned above for performance evaluation. The benchmarks namely TPC-B [4], and TATP [3] are run on PostgreSQL 9.0.0 with default settings, e.g., the page size is 8KB. Dataset size of both TPC-B and TATP are 2GB. The MLK is a record of the page accesses of making Linux kernel 2.6.27.39. We use a tool named strace to monitor theses processes and obtain the disk access history. Specification on the traces was given in Table 3.

Table 3. Specification on the Traces

Filename	Page Number (10^3)	Reference Number (10^6)	Write Ratio
MLK	97.2	27.2	0.43%
TATP	135.1	2.5	4.59%
TPC-B	35.1	10.7	19.46%

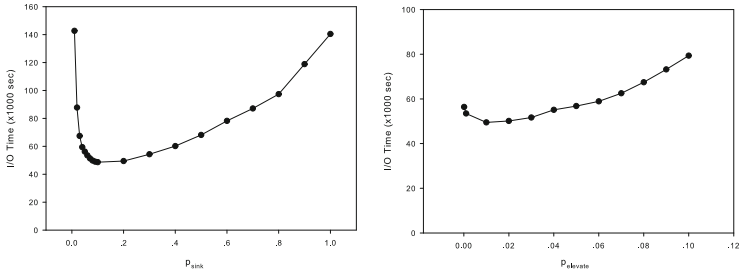
In our experiment, the *total I/O time* is used as the primary metric to evaluate the performance. We employ The samsung SSD (64GB, 470 series) in our experiment. We obtain its access latency by testing. Access latency of hard disk is obtained from paper [15]. The parameters used in our experiments are listed in Table 4, in which $Flashsize/Pages$ denotes the ratio between flash and dataset size. We fixed the main memory size to the 1% of the dataset size.

Table 4. Experimental Parameters

Parameter	Value
C_r, $C_w(\mu s)$	271, 803 (for SSD) 12700, 13700 (for hard disk)
$Flashsize/Pages$	1.25%, 2.5%, 5%, 10%, 20%
$p_{elevate}$	0.001,0.01, 0.015,0.02,...,0.1
p_{sink}	0.01, 0.1,0.2,...,0.9

5.2 Parameter Tuning

To begin with, we inspect the effect of parameters on performance of algorithms. We illustrate how the performance of HyPro varies with p_{sink} in Figure 5 (a), along with the values of C_{sinkf} and C_{sinkd}. The HyPro achieves the best performance at around $p_{sink} = 0.15$. p_{sink} reflects the chance for a page evicted from main memory to be stored into flash. A low p_{sink} will result in a poor utility of flash memory. On the contrary, an excessive high p_{sink} would incurs great exchange cost between main memory and flash. Thus, our approach achieves the best performance with proper p_{sink}.



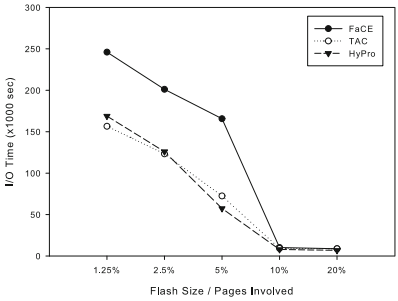
(a) p_{sink} tuning on HyPro (b) $p_{elevate}$ tuning on HyPro

Fig. 5. Parameters Tuning

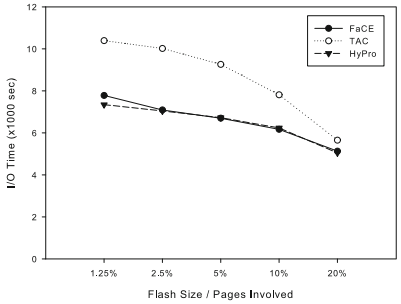
The performance of our approach also significantly varies with the $p_{elevate}$, as shown in Figure 5 (b) , and the minimum I/O time is reached when it locates between 0.01 and 0.02. As the $p_{elevate}$ controls whether a page on flash should be elevated, large $p_{elvevate}$ will cause more exchanges between main memory and flash, which may deteriorates the whole performance, while no exchange will cause pages on flash has no opportunity to get into main memory (corresponding to the case $p_{elevate} = 0$). Other testings also show that the best parameter is often around 0.02 and 0.2, and thus, we adopt these values as the initial value and use dynamic parameter tuning in the following experiments.

5.3 Comparison with Other Approaches

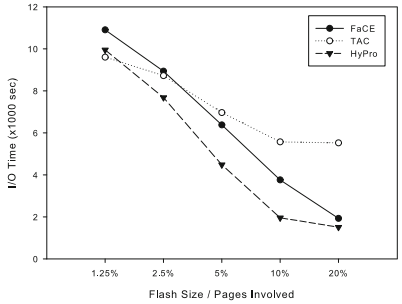
In this part, we compare HyPro with FaCE and TAC. We test extensive configurations, but only shows some results here in Figure 6 due to the space limitation. In the following results, the main memory is 1% of the total workload size and the flash size varies from 1.25% to 20%. Our approach shows similar or better performance compared with FaCE and TAC approach. Our approach can reduce upto 50% of the total I/O time against other competitors. In MLK trace, FaCE performs the worst, since many pages in MLK are accessed only once, but the FaCE still cache these pages in flash memory. On TATP, HyPro and Face has similar performance and better than TAC. Because the TATP is not a very stable access pattern. FaCE and HyPro can adapt themselves to the workload changes quickly, but TAC needs a rather long time to learn this change. On TPC-B trace TAC has the best performance when the flash size is low, this is because the temperature-based hot detection can accurately discover the hottest pages, which has superiority when the cache size is low. However, When the flash size is large, the performance of TAC degrades. This phenomenon is partially because precisely hot page detection is not necessary for a larger cache size, and partially because of the write through cache design, as the write ratio is very high in TPC-B according to Table 3. The HyPro can adapt itself to the flash size enlargement and shows a good performance in all the cases.



(a) Low-end SSD performance on MLK trace



(b) High-end SSD performance on TATP trace



(c) Low-end SSD performance on TPC-B trace

Fig. 6. I/O performance comparison on benchmark traces

6 Conclusion

In this paper we propose a novel stochastic approach for flash based hybrid storage system management named HyPro. Different from the existing deterministic models, HyPro controls the data migration between devices using two probabilities. One probability describes the chance of which one page will be kept in main memory after accessed from flash. Another probability is adopted to determine the place where a page should be put after evicted from main memory. By doing this, Hypro can achieves lower hard disk access with a little exchange overhead increment on flash writes. We also developed an approach to determine the probabilities based on cost analysis. The experiments show that HyPro outperforms other competitors.

Acknowledgments. This research was supported by the grants of Natural Science Foundation of China (No. 61272155, 61073019) and MIIT grant 2010ZX01042-001-001-04.

References

1. http://windows.microsoft.com/en-US/windows-vista/products/features/performance
2. Momentus XT Solid State Hybrid Drives, http://www.seagate.com/www/en-us/products/laptops/laptop-hdd/
3. Telecom Application Transaction Processing Benchmark, http://tatpbenchmark.sourceforge.net/index.html
4. TPC Benchmark B (TPC-B), http://www.tpc.org/tpcb/
5. Bisson, T., Brandt, S.A., Long, D.D.E.: A hybrid disk-aware spin-down algorithm with I/O subsystem support. In: IPCCC, pp. 236–245 (2007)
6. Canim, M., Bhattacharjee, B., Mihaila, G.A., Lang, C.A., Ross, K.A.: An object placement advisor for DB2 using solid state storage. PVLDB 2(2), 1318–1329 (2009)
7. Canim, M., Mihaila, G.A., Bhattacharjee, B., Ross, K.A., Lang, C.A.: SSD bufferpool extensions for database systems. PVLDB 3(2), 1435–1446 (2010)
8. Chen, S.: Flashlogging: exploiting flash devices for synchronous logging performance. In: SIGMOD Conference, pp. 73–86 (2009)
9. Debnath, B.K., Sengupta, S., Li, J.: Flashstore: High throughput persistent key-value store. PVLDB 3(2), 1414–1425 (2010)
10. Debnath, B.K., Sengupta, S., Li, J.: Skimpystash: RAM space skimpy key-value store on flash-based storage. In: SIGMOD Conference, pp. 25–36 (2011)
11. Do, J., Zhang, D., Patel, J.M., DeWitt, D.J., Naughton, J.F., Halverson, A.: Turbocharging DBMS buffer pool using SSDs. In: SIGMOD Conference, pp. 1113–1124 (2011)
12. Jiang, S., Zhang, X.: LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. In: SIGMETRICS, pp. 31–42 (2002)
13. Kang, W.-H., Lee, S.-W., Moon, B.: Flash-based extended cache for higher throughput and faster recovery. Proc. VLDB Endow. 5(11), 1615–1626 (2012)
14. Koltsidas, I., Viglas, S.: Flashing up the storage layer. PVLDB 1(1), 514–525 (2008)
15. Lee, S.-W., Moon, B.: Design of flash-based DBMS: an in-page logging approach. In: SIGMOD Conference, pp. 55–66 (2007)
16. Luo, T., Lee, R., Mesnier, M.P., Chen, F., Zhang, X.: hStorage-DB: Heterogeneity-aware data management to exploit the full capability of hybrid storage systems. CoRR abs/1207.0147 (2012)
17. Megiddo, N., Modha, D.S.: ARC: A self-tuning, low overhead replacement cache. In: FAST (2003)
18. Ou, Y., Härder, T.: Trading memory for performance and energy. In: Xu, J., Yu, G., Zhou, S., Unland, R. (eds.) DASFAA Workshops 2011. LNCS, vol. 6637, pp. 241–253. Springer, Heidelberg (2011)
19. Wu, X., Reddy, A.L.N.: Managing storage space in a flash and disk hybrid storage system. In: MASCOTS, pp. 1–4 (2009)