HDCat: Effectively Identifying Hot Data in Large-scale I/O Streams with Enhanced Temporal Locality

Jiahao Chen[1], Yuhui Deng[1(✉),2], and Zhan Huang[1]

[1] Department of Computer Science, Jinan University, Guangzhou, P.R.China, 510632
[2] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R.China, 100190
cm2243@foxmail.com, tyhdeng@jnu.edu.cn, thz@jnu.edu.cn

Abstract. Hot data is very important for optimizing modern computer systems. For example, the identified hot data can be employed to extend the lifespan of flash memory. However, it is very challenging to effectively identify hot data with low memory consumption and low runtime overhead. This paper proposes a Hot Data Catcher (HDCat) which can effectively identify hot data in large-scale I/O streams by leveraging enhanced temporal locality. HDCat only maintains a hot data queue and a candidate hot data queue to record the data access pattern by tracking limited data set, thus effectively reducing the memory consumption. Furthermore, HDCat adopts a D-bit counter and a recency-bit to leverage both the frequency and recency contained in the data stream. Additionally, HDCat can significantly reduce the conversion between hot data and cold data. Real traces are used to evaluate the proposed approach. Experimental results demonstrate that HDCat significantly outperforms the state-of-the-art Multi-hash algorithm and the two-level LRU algorithm.

Keywords: Hot Data; Identifying Hot Data; Large-scale; Temporal Locality; Hot Data Identification Algorithm

1 Introduction

Hot data is very important for optimizing modern computer systems. The traditional hot data identification algorithms simply recorded the occurrences of the data items and ignore the volume of the corresponding data set. Therefore, most of the traditional approaches employed to identify hot data incured large memory consumption [4] or a high runtime overhead [1]. Additionally, these approaches did not consider the temporal locality which has a significant impact on the hot data identification [2] [12]. This is because the recently accessed data is more likely to be accessed again in the near future. In order to overcome this problem, Hsieh et al. [8] proposed a multiple hash function framework to identify hot data. This algorithm used multiple hash functions and a Boolean filter to capture the frequency information of data access pattern. Furthermore, it employed a counter to accurately capture the frequency information. However, the exponential decay mode adopted in the algorithm (i.e. every time all the counters are reduced by half) made it hard to capture the temporal locality of the data access pattern.

Hot data identification methods can be applied in different scenarios. Caching is typical scenario [5] [10] [7]. By caching hot data in advance, we can significantly improve the corresponding system performance.When the hot data is transferred to cold data, in order to make room for the other data to cache, it has to write the data back to flash memory. Due to the characteristics of flash memory that it cannot be updated in place, the data from memory cannot be directly over write the original data, and it will be written to another clean s-torage space. The old data should erase in advance, and the erasing operation in flash memory is in block units to implement, which makes the data stored in the same block need to be copied to other clean data space. If the conversion between hot and cold data arises frequently, it will greatly increase the number of erasing operation in flash memory [13] [9]. However, the flash memory can be written out by a finite number of erasing operation, thus it will greatly reduce the service life of flash memory [15] [6]. In addition, erasing operation (500μs) [14]was expensive in flash memory compared with the read (25μs) and write (200μs) operation. Thus, frequent conversion between hot and cold data will lead to frequent erasing operation as well as an unpredictable delay to system. In general, reducing the number of conversion between hot and cold data will help reduce the number of erase operation on flash memory as well as prolong its service life and prevent unpredictable delay in system. Although there have been a lot of researches on flash memory and its optimization, seldom from the perspective of hot data identification algorithm to prolong the service life of the flash memory and improve its performance.

This paper proposes a Hot Data Catcher (HDCat) to effectively identify hot data by leveraging a recency-bit and a D-bit counter. The key idea is that: 1. Update the D-bit counter of item according to its recency bit which will lead to a result that the recently access data (recency bit=1)will increase its heat value a bit faster than the item has not been accessed for a period of time(recency bit=0). 2. Filtering mechanism based on the recency information, when the algorithm needs to eliminate an item, it first chooses the lowest counter item with a negative recency bit. In this case, an item with a positive recency bit will not be eliminated even though its counter is lower. 3. Our trembling mechanism and the two level structure, these feature are in order to reduce the run time overhead and the memory consumption respectively. Experimental results demonstrate HDCat that can effectively improve the accuracy of hot data identification, while maintaining low memory overhead and low runtime overhead, and significantly reducing the conversion between hot data and cold data.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents the design and operation of the hot data catcher. Section 4 shows the experimental results. Finally, Section 5 summarizes and concludes the paper.

2 Related work

2.1 Bloom filter

The main goal of bloom filters (BFs) is recording information with a low s-torage overhead. For BFs, space efficiency is a very important factor. Therefore, it often maximizes the space efficiency at the expense of correctness. Even if an element does not belong to a given set, BFs may also give a wrong positive answer, and judge the element included in the set. This is called false positive. However, the basic BFs do not cause false negative problem, if we give an element belonging to a set, it will generate a right answer. We can modify some parameters (i.e., BF size, the number of hash functions and the number of unique elements) of BFs to reduce the probability of false positive.

2.2 D-bit counter

The information a simple Boolean record is too limited, and cannot meet the demand of information we have on many occasions, but the use of plastic variables to record data will takes up too much storage space, a D-bit counter will be an acceptable compromise.

D-bit counter is a D bits array, which can record the data range from zero to $2^D - 1$.We first set all bits in counter to 0, whenever the corresponding element appears, the entire counter value is incremented by 1.When the entire counter value exceeds$2^D - 1$, we will stop adding 1 operation. D-bit counter also includes a B-most Significant Bits, which we call threshold. When all the bits after the B-most Significant Bits are 0, we can judge the frequency of data does not exceed the threshold. Otherwise, the data will be regard as exceed the given threshold. In general, D-bit counter is a structure that can provide sufficient accuracy as well as space efficiency. The algorithm we proposed and the famous Multiple Hash Function have employed this structure. Subsequent chapters will introduce more detail.

In hot data identification, we are concerned about the data access pattern but not the detail access information. For example, we need to know whether a data accessed frequently, or to be exact, we want to know whether the occurrence of this data exceeds a given threshold. However, once we know that a data access frequency exceeds the thresholds, we generally do not care about the specific number of request. We do not need to use an integer variable to record the complete access information. In this scenario, D-bit counters satisfy our needs well, and provide space efficiency.

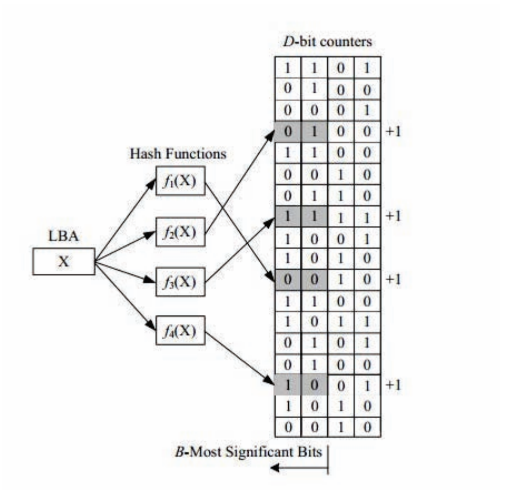


Fig. 1. Multiple Hash Function Framework. HereD = 4 and B = 2.

2.3 Hot Data identification algorithms

In order to decrease the memory consumption of hot data identification, Chang et al. [1] proposed a two-level LRU algorithm (TLL) including a hot data list and a candidate hot data list. Both two lists have a fixed length and are LRU lists. Whenever a request arrives, TLL checks if the corresponding LBA is in the hot list. If it hits the hot list, then the data will be regarded as a hot data. Otherwise, the data is treated as cold data. If the data is not in the hot list but in the candidate list, then the data is promoted to the hot list. If the data is not in both lists, they are inserted to the candidate list. This two-level LRU algorithm has better space efficiency than FMS, because it only needs to record the access information of hot data and the candidate data, instead of recording all of the data access information. Unfortunately, it has another problem. Its performance relies heavily on the length of the two lists. In other words, a short list can reduce memory overhead, but it would improperly degrade hot data to the candidate list. This will decrease the accuracy of identification. Another disadvantage of this algorithm is that, it incurs a high run-time overhead when running the algorithm.

Recently, Hsieh et al. [3] proposed another algorithm called multiple hash function framework (MHF) to identify hot data (Fig. 1).This MHF adopts multiple hash functions and a D-bit counter. MHF records the data access information by incrementing the corresponding counters. It updates the data access information by periodically dividing the counter by 2. If a bit after the B-most significant bits is set to 1, the corresponding counter returns 1. Otherwise, it returns 0. Similar to the method of bloom filters, MHF also adopts K independent hash functions. If all K-bit positions (from the k-hash functions) are set to 1, the data is regard as a hot data. For example, as shown in figure 1, the assumption is that the algorithm uses 4-bit counters and adopts 2 as its most significant

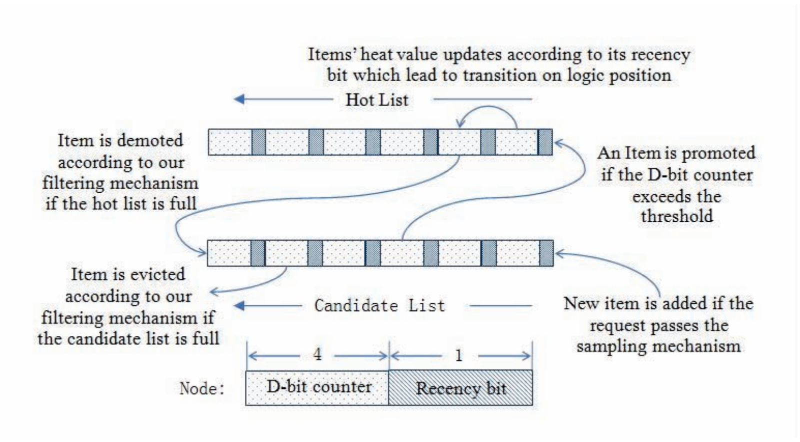

Items' heat value updates according to its recency bit which lead to transition on logic position
Hot List
Item is demoted according to our filtering mechanism if the hot list is full
An Item is promoted if the D-bit counter exceeds the threshold
Item is evicted according to our filtering mechanism if the candidate list is full
Candidate List
New item is added if the request passes the sampling mechanism
4
1
Node:
D-bit counter
Recency bit

Fig. 2. D-bit counters with Recency bit in Two-level structure

bit, then 4 will be its hot threshold. In this example, we map the address of the data X with 4 hash functions (f1, f2, f3, f4) and we then get 4 values: 0010, 0100, 1111, 1001. Here only the counter corresponding to f2 returns 0, but it is enough to judge data X as a cold data. Due to the ageing mechanism, this algorithm decreases all the counters by a half with 1-bit right shifting after a specified time interval. Compared with other algorithms, MHF achieves a relatively low memory overhead and run-time overhead.However, it does not catch recency information appropriately due to its exponential decrement of all LBA counters.

3 HotData Catcher

3.1 Overview of Hot Data Catcher

As shown in Fig. 2. HDCat consists of a hot list and a candidate hot list to identify hot data. Each item on the list contains a recency bit and a D-bit counter. Initially, both lists are empty. All the data items are treated as cold data. When a new data item arrives, HDCat first checks whether it is on the two lists. If any list contains the data item, the corresponding value of the D-bit counter will be increased. If a data item is on the candidate list and its D-bit counter is bigger than a given hot threshold, this data item will be promoted to the hot list. If the hot list is full, a data item on the hot list will be demoted to the candidate hot list by leveraging a filtering mechanism. If the two lists do not contain the new data item, a data item on the candidate list will be evicted and the new data item will be recorded on the candidate hot list.

Each item on the list contains a recency bit and a D-bit counter. The D-bit counter is used to store the access frequency. It is in the range of 0 to $2^D - 1$. The recency bit is adopted to identify whether the associated item is recently accessed. Both the D-bit counter and the recency bit are initialized to zero. If the recency bit is 0, it indicates that the item has not been visited for certain a period of time. When an item on the list is accessed, the D-bit counter will be increased by 1, and the recency bit will be changed to 1. This indicates that the corresponding item has just been visited. If the recency bit is 1, and the associated item has been accessed within a short period of time, we increase the associated D-bit counter by 2. If the value of a D-bit counter exceeds $2^D - 1$, we cannot get the exact access frequency. However, HDCat only needs to know whether the D-bit counter exceeds a given threshold but not an exact number of occurrences. This feature makes D-bit obtain high accuracy of identification with a low storage overhead. If the D-bit counter overflows, a freezing approach [14]is adopted to handle this issue. The approach does not increase the counter value any more even if the corresponding data is access again. The D-bit counters are not simply storing the occurrences of data, but the heat value which represents frequency as well as recent information of the access pattern. According to the mechanism of HDCat, even if the occurrences of two data items are equal, but one focuses on the past, another one focuses on the recency, the two data items will get different heat value. This is reasonable for a real scenario of hot data identification. Additionally, we believe that a data item continuously accessed for a short period of time is a better candidate of hot data than a data item referenced within a long period of time. This scenario will generate different heat values for the two data items when using HDCat. For example, if a data item has been visited repeatedly over a short period of time, this data is more likely to become a hot data. Therefore, the recency bit of the data item will be set to 1 and its heat value will grow fast. HDCat employs an aging mechanism to handle this issue where the D-bit counter does not simply increase with the increase of its occurrence, but becomes less and less as time goes by. Algorithm 1 summarizes the working process of HDCat using pseudo-code. When a new data item arrives, there are three different scenarios which handle the new data item differently. (1) If the data item is on the hot list, the D-bit counter of data item is increased by 1. If the recency bit of the data item is 0, it will be changed to 1. Otherwise, if the recency bit of the data item is already set to 1, the D-bit counter will be added 2 indicating that this data item has recently been accessed. (2) If the new data item is not on the hot list but on the candidate hot list, HDCat will deal with the data item like step one. Furthermore, if the D-bit counter of the data item exceeds a given threshold, this date item will be promoted to the hot list. If the hot list is full, one data item on the list will be demoted using the filtering mechanism to make space. (3) If the new data item is not on the two lists, HDCat will perform a sampling mechanism. If it passes the sampling mechanism, the data item will be inserted to the candidate hot list. Otherwise, the data item will be discarded.

3.2 Design of Hot Data Catcher

In order to effectively identify hot data, HDCat involves three important components including a filtering mechanism, a sampling mechanism and an aging

```
Algorithm 1: Hot Data Catcher
Data: Request for item
Result: Hot or Cold data classification of the itme
1  begin
2      A Request for an item x is issued
3      if HotList Hit then
4          Increase its coounter according to its recency bit
5          Set the recency bit to 1
6          Classify x as hot data
7      else
8          //HotList Miss
9          if CandidateList Hit then
10             Increase its counter according to its recency bit
11             Set the recency bit to 1
12             if Counter greater than HotThreshold then
13                 //promote to HotList
14                 if HotList is not full then
15                     Promote x to the HotList
16                 else
17                     //Need to evict item
18                     find the evict data y with our filtering mechanism
19                     demote y to the CandidateList
20         else
21             //CandidateList Miss
22             if Pass the sampling test then
23                 if CandidateList is full then
24                     find the evict item y with our filtering and remove it
25                 Insert x into the CandidateList
26             else
27                 Skip further processing of the x
```

mechanism. The data structures used to record data access information include one recency bit and a D-bit counter, where D is equal to 4 in our experiment. The recency bit divides the data of list Lx ($x \in 0, 1$, L0 represents the hot list, L1 represents the candidate hot list) into two sets S1 and S0 ($S1, S0 \subseteq Lx\ and\ S1 \bigcup S0 = Lx$). S1 represents a data set that the recency bit of the data is 1 and the data has been recently visited. S0 represents the remaining data.

Filtering mechanism When Lx is full and a new data item has to be inserted into this list, it invokes the filtering mechanism. The filtering mechanism can be divided into two parts. The first part is selecting the data item that has the minimum value of the 4-bit counter in S0 and removing it. This indicates that if $S0 \notin \emptyset$, the data item s will be removed from $S0\{s \mid (\forall v)s, v \in S0 \wedge s.counter \leq v.counter\}$. If there is no data item with the recency bit setting to 0, the second part will select a data item from S1. Similarly, the data item with the minimal counter value is selected. This means, if $S0 \in \emptyset$, the data item s will be removed from $S1\{s \mid (\forall v)s, v \in S1 \wedge s.counter \leq v.counter\}$. After finishing the filtering process, the recency bits of all data items are changed to 0 except the newly added one. The data item removed from hot list will be inserted into the candidate hot list. If the candidate hot list is full, we will perform the filtering mechanism on the candidate hot list with the same process. This is the conversion between hot and cold data.

Since the filtering mechanism has to traverse the whole list, this run-time overhead may lead to a performance degradation of the system. In order to alleviate this problem, the filtering mechanism locates a data item that the associated heat value (the value of 4-bit counter) is less than a threshold instead of finding the data item with the minimum heat value. Once the data item is located, the traversal process will be stopped.This applies to the candidate hot list as well. In the worst case, traversing the whole list may not be able to locate a qualified data item, we take the data item which has the minimum heat value across the whole list. Since we only consider the data items with high heat value, this simple optimization can significantly reduce the runtime overhead with a negligible impact on the accuracy of hot data identification.

Sampling mechanism If a forthcoming data item is not on the both lists, but it would be inserted to the candidate hot list, a sampling mechanism is triggered to handle this scenario. The mechanism puts the forthcoming data item on the candidate hot list with a certain probability. The goal of this mechanism is avoiding the frequent conversion between hot data and cold data, thus further reducing the runtime overhead of the algorithm. For example, we can insert a new data item to the candidate hot list with a 50% probability. This mechanism will not change the probability that a frequently accessed data item is inserted to the candidate hot list. Since the data is frequently accessed, the opportunity of passing this sampling is also bigger than other data. However, this simple sampling mechanism helps HDCat discard those infrequently accessed data at a very early stage. Therefore, it reduces not only memory consumption, but also computational overhead. If a data item passes the sampling mechanism, the data will be inserted to the candidate hot list. If the candidate list is full, HDCat will employ the filtering mechanism to remove a data item from the candidate hot list. Combing with the D-bit counter and recency bit, the HDCat can achieve very high accuracy of hot data identification with low runtime overhead and memory consumption.

Aging mechanism HDCat employs an aging mechanism to update the data access information. It cuts all the values of the D-bit counters by half within a fixed period of time, thus updating the data access information. Therefore, even if one data item is frequently accessed in the past, as long as it is no longer

Table 1. System Parameters and Values

System Parameters	HDCat	MHF	TLL
Number of Counter	2^{12}	2^{13}	2^{12}
Counter Size	4	4	N/A
Decay	2^{12}/(1-20%)	2^{13}/(1-20%)	2^{12}/(1-20%)
Number of Hash Function	N/A	2	N/A
Hot Threshold	4	4	N/A
Number of Levels	2	N/A	2

Table 2. Workload Characteristics

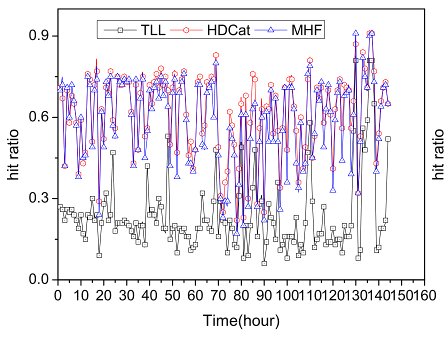
Workloads	Total Requests	Trace Features	(Read:Write)	Total Request Blocks
hm	4,602,527	Hardware monitoring	R:(43.4%) W:(56.6%)	82,310,381
wdev	1,326,264	Test webserver	R:(17.3%) W:(82.7%)	23,727,666
rsrch	1,655,022	Research projects	R:(16.3%) W:(83.7%)	27,636,758

frequently visited over a certain period of time, this data item will eventually become a cold data and be demoted from the hot list. The aging mechanism of HDCat employs M arrays, each array consists of a 4-bit BF as the counter. Its decay period is N. It indicates that after N requests, the aging mechanism will be invoked. This decay period must ensure that the hash table sizing M can accommodate all the N requests within this period, where $N \leq M/(1-R)$, R represents the hot ratio of the data access pattern, R=20% [8]. As long as we have determined the capacity of the M hash table arrays, we can find out the corresponding decay period N. The decay period of HDCat is set as N that is defined as 4096 in our experiment.

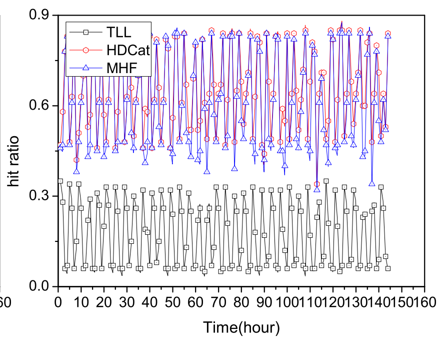
4 Evaluation

4.1 Evaluation Environment

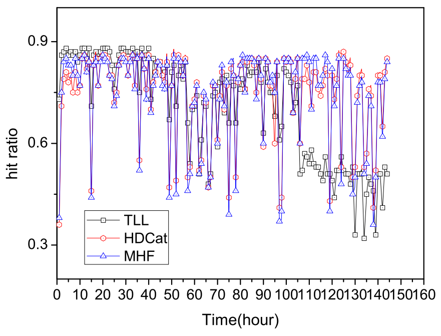
In order to evaluate the performance behavior of HDCat, we designed two state-of-the-art schemes including a Multi-hash function (MHF) [8]and a two-level LRU (TLL) scheme [1]for comparison. Table 1 summarizes the parameters used by the three schemes. For a fair evaluation, we adopt the same decay interval (4,096 write requests) and an identical aging mechanism for all the three schemes. Three real traces are used to evaluate the schemes. The traces are collected from the core servers in Microsofts data centre at block level by using event tracing for Windows [11]. The time length of the three traces are all 144 hours. Table 2 shows the characteristics of three traces, where HM tracks the data access pattern of hardware monitoring, WDEV collects the data accesses of a test web



(a) Hm



(b) Wdev



(c) Rsrch

Fig. 3. Hit Ratio of three algorithms with different traces

server and RSRCH logs the disk activities of a research projects. Each record in the traces includes timestamp, request type, data address offset, data block size. Each request in the traces consists of several sectors which are equal 512 Bytes. For example, if the offset field of a request is 1001, and the size is 1024, this indicates that the request can be divided into two sector requests, and the block address of the actual request are 1001 and 1002, respectively.

In our experiments, we replay the whole traces in terms of the timestamp. Furthermore, a typical cache simulator is designed to evaluate the schemes. Hit ratio is employed as a very important metric to devaluate the schemes. In the condition that each algorithm can cache the same amount of data blocks, higher hit ratio indicates better performance of the corresponding scheme. In addition to the hit ratio, we also compare the conversions between hot and cold data across the three schemes.

4.2 Eexperimental results

Evaluating hit ratio Fig. 3 shows the hit ratio of three schemes when using three different traces, where X axis represents time length, and Y axis indicates

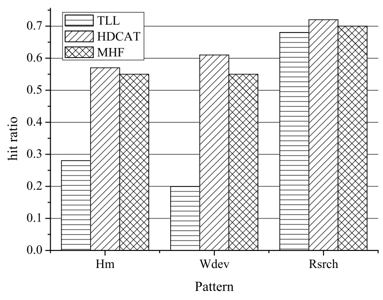
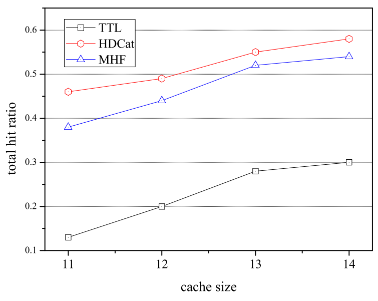
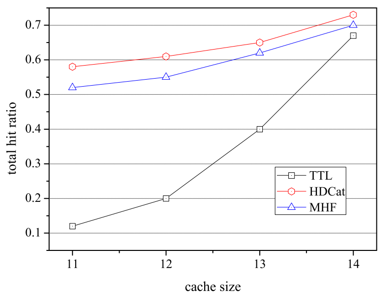


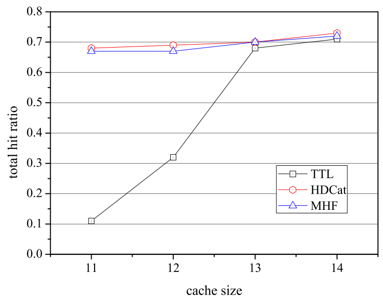
Fig. 4. Total hit ratio of three algorithms with three traces



(a) Hm



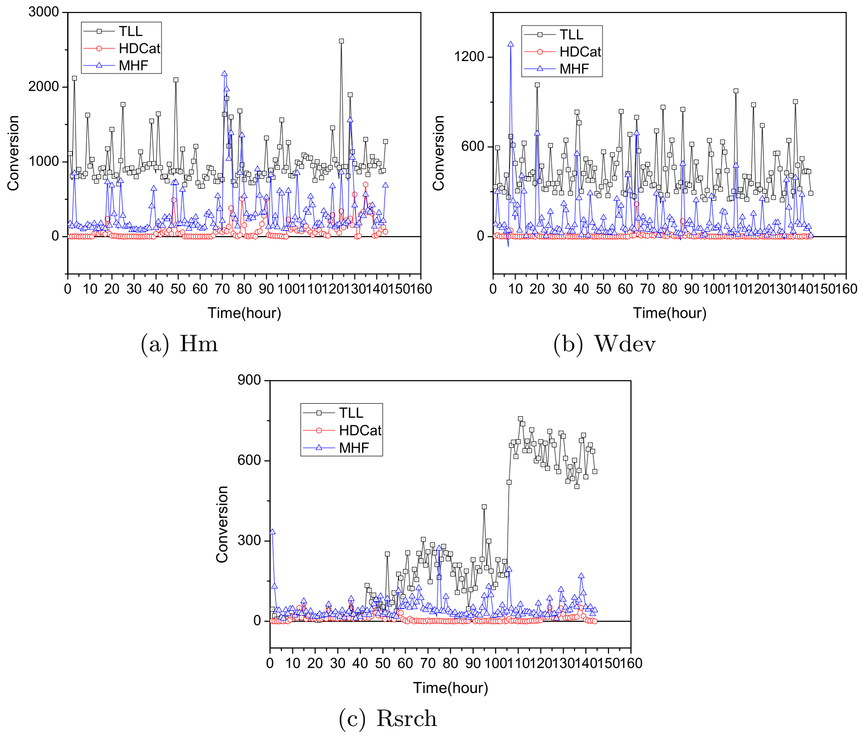
(b) Wdev



(c) Rsrch

Fig. 5. Total Hit Ratio under different cache capacity

the hit ratio. The experimental results show that both HDCat and MHF have a much higher hit ratio than that of TLL. This is because the accuracy of hot data identification depends on the length of the two lists when using TLL. Since we use a small Cache capacity in our experiments, the length of lists used to record



(a) Hm (b) Wdev (c) Rsrch

Fig. 6. Frequency of hot/cold data conversion

data items is limited. Furthermore, From the figure, we can also see that the hit ratios of TLL are lagged far behind that of HDCat and MHF when using HM trace and WDEV trace. This is because HM and WDEV have very large address space, which results in very low spatial locality. Therefore, the performance of TLL is tightly correlated with the spatial locality contained in the traces. Fig. 4 summarizes the total hit ratio of the three schemes with three different traces. It indicates that HDCat significantly outperforms MHF and TLL.

Fig. 5 shows the impact of cache capacity on the hit ratio of three different schemes, where X axis represents the size of cache capacity. (e.g. number 12 represents that the size of cache is 2^{12}=4096.). It demonstrates that TLL performs worst, and the hit ratio of TLL grows significantly with the increase of Cache capacity. This confirms that the performance of TLL algorithm is very dependent on its list length (cache size). Although both HDCat and MHF achieve very good performance, Fig. 5 also implies that HDCat achieves a better performance than MHF across different cache capacity.

Evaluating the conversion of hot/cold data Even with the same hit ratio, different algorithms may still have a different read/write frequency due to the

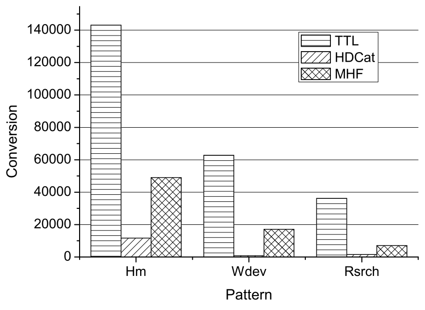


Fig. 7. Total conversion of three algorithms with three traces

conversions between hot data and cold data. Fig. 6 shows the number of conversions between hot data and cold data when using different schemes and different traces, where the X axis represents time length (144 hours in our experiment) of replaying traces, and the Y axis represents the number of conversions. The statistics are calculated every one hour. The figures demonstrate that HDCat incurs the minimal number of conversions across the three traces. This feature can be leveraged by many applications. For example, if HDCat is deployed for flash memory, it will effectively reduce the number of write operations, thus extending the effective life span of the flash memory without reducing its hit ratio. Furthermore, HDCat is more stable than MHF and TLL.We also summarizes the total conversions of the three schemes with three different traces.In Fig. 7 we can see that HDCat significantly outperforms MHF and TLL.

5 Conclusion

This paper proposes and designs a hot data identification algorithm called HDCat by leveraging the combination of a D-bit counter and a recency bit.Real traces are employed to evaluate HDCat against two state-of-the-art schemes including a multi-hash function method and a two-level LRU approach. Experimental results demonstrate that HDCat can accurately capture the temporal locality of data access patterns and achieve a high hit ratio with low cache capacity and runtime overhead. Furthermore, HDCat significantly reduces the number of conversions between hot and cold data, thus decreasing the number of write operations. Therefore, HDCat is a very good candidate method for optimizing the performance the reliability of flash memory. Furthermore, we believe HDCat can be applied to many scenarios to optimize the computer systems.

6 Acknowledgments

This work is supported by the National Natural Science Foundation (NS-F) of China under Grant (No. 61572232, and No. 61272073), the key program of Natural Science Foundation of Guangdong Province (No.S2013020012865), the Open Research Fund of Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences (CARCH201401), and the Fundamental Research Funds for the Central Universities, and the Science and Technology Planning Project of Guangdong Province (No. 2013B090200021).

References

1. Chang, L.P., Kuo, T.W.: An adaptive striping architecture for flash memory storage systems of embedded systems. IEEE Real-time Embedded Technology Applications Symposium (2002) 187 – 196
2. Chang, L.P., Kuo, T.W.: Efficient management for large-scale flash-memory storage systems with resource conservation. ACM Transactions on Storage (2005) Vol.1(4):381–418
3. Chang, L.P., Kuo, T.W., Lo, S.W.: Real-time garbage collection for flash-memory storage systems of real-time embedded systems. ACM Transactions on Embedded Computing Systems (2004)
4. Chiang, M.L., Paul, C.H.L., Chang, R.C.: Managing flash memory in personal communication devices. In: Proceedings of the 1997 International Symposium on Consumer Electronics (ISCE'97. (1997) 177–182
5. Debnath, B., Subramanya, S., Du, D., Lilja, D.J.: Large block clock (lb-clock): A write caching algorithm for solid state disks. In: Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on. (2009) 1 – 9
6. Deng, Y.: What is the future of disk drives, death or rebirth? Acm Computing Surveys (2011) Vol.43(3):194–218
7. Deng, Y., Wang, F., Na, H.: Eed: Energy efficient disk drive architecture. Information Sciences (2008) Vol.178(22):4403–4417
8. Hsieh, J.W., Chang, L.P., Kuo, T.W.: Efficient identification of hot data for flash memory storage systems. ACM Transactions on Storage (TOS) TOS Homepage (2006) Vol.2:22–40
9. Jo, H., Kang, J.U., Park, S.Y., Kim, J.S., Lee, J.: Fab: flash-aware buffer management policy for portable media players. IEEE Transactions on Consumer Electronics (2006) Vol.52(2):485 – 493
10. Kim, H., Ahn, S.: Bplru: A buffer management scheme for improving random writes in flash storage. FAST (2008) 239–252
11. Narayanan, D., Donnelly, A.: Write off-loading: practical power management for enterprise storage. Trans. Storage (2008) Vol.4(3)
12. Park, D., Debnath, B., Du, D.: Cftl: A convertible flash translation layer adaptive to data access patterns. In: in SIGMETRICS. (2010) 365–366
13. Park, S.Y., Jung, D., Kang, J.U., Kim, J.S., Lee, J.: Cflru: a replacement algorithm for flash memory. In CASES 06: Proceedings of the 2006 international conference on Compilers, architecture (2006) 234–241
14. Parkz, D., Nam, Y.J., Debnath, B., Du, D.H.C., Kim, Y., Kim, Y.: An on-line hot data identification for flash-based storage using sampling mechanism. ACM SIGAPP Applied Computing Review (2013) Vol.13(1)
15. Zhang, L., Deng, Y., Zhu, W., Zhou, J., Wang, F.: Skewly replicating hot data to construct a power-efficient storage cluster. Journal of Network and Computer Applications (2015) 168–179