ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.11, No.11, 1–7

Frequency-based NCQ-aware disk cache algorithm

Young-Jin Kim[a)]

Ajou University,

206, World cup-ro, Yeongtong-gu, Suwon-si, Gyeonggi-do 443-749, Republic of Korea

a) *youngkim@ajou.ac.kr*

Abstract: In SATA hard disks, native command queuing (NCQ) and a cache can play important roles in boosting the performance. However, research on cache algorithms which can exploit the benefits fully from NCQ has been seldom reported till now. In this paper, we propose a novel cache algorithm which combines disk access time and access frequency in a more NCQ-friendly way to enhance the I/O performance. Real trace-driven simulations show that the proposed algorithm improves the overall I/O performance by up to 39.2 and 19.1 percent over LRU and a prior access time-aware cache algorithm, respectively.

Keywords: SATA hard disk, disk cache, native command queuing, access time, frequency, performance

Classification: Storage technology

References

[1] Intel Corporation and Seagate Technology: joint white paper (2003) http://www.seagate.com/docs/pdf/whitepaper/D2c_tech_paper_intc-stx_sata_ncq.pdf.

[2] K. Grimsrud and H. Smith: *Serial ATA storage architecture and applications* (Intel Press, 2007) 21.

[3] A. Arfan, Y.-J. Kim and J. B. Kwon: IEICE Electron. Express **9** (2012) 1707. DOI:10.1587/elex.9.1707

[4] J.-U. Kang, H. Jo and J.-S. Kim: EMSOFT (2006) 161.

[5] Y.-J. Kim, S.-J. Lee, K. Zhang and J. Kim: IEEE Trans. Consum. Electron. **53** (2007) 1469. DOI:10.1109/TCE.2007.4429239

1 Introduction

Hard disks have been used as a major storage device in most computing systems including PC and servers for decades. Since hard disk contains some mechanical components such as a rotating platter and a head to read (write) data from (to) the platter, whose speeds are rather slower than those of electronic components, it is considered a bottleneck in the aspect of the overall performance, compared to memory and processor.

Hence, there have been a lot of research on improving the performance of hard disk. The most important is that the Serial ATA (SATA) has been suggested and used to boost the speed of a hard disk [1, 2]. Native command queuing (NCQ) is one of main features of SATA and is beneficial in enhancing

DOI: 10.1587/elex.11.20140363
Received April 17, 2014
Accepted April 21, 2014
Publicized May 14, 2014
Copyedited June 10, 2014

the I/O performance by reordering the requests which should be served by a hard disk.

Access time indicates the elapsed time in accessing a specific block on a hard disk or a cache. In detail, access time in a SATA hard disk consists of seek time to position the actuator, rotational latency time to wait for the data to rotate under the head, and data transfer time. Since NCQ reschedules the request order of requests staying in a queue, seek time and rotational latency will be affected much. Thus, the access time depends on the request order and fluctuates highly.

In the meanwhile, in order to improve the performance of a hard disk, we usually employ a disk cache to it. Since a cache can serve some valuable data prior to the disk platter, the data will be processed with a faster service time. For the last decades, the least recently used (LRU) algorithm has been mainly employed to manage a disk cache. Unfortunately, LRU has no idea of which request should be serviced first or not in any other aspects except temporal locality. Thus, LRU seems difficult to co-work with NCQ in a synergistic manner for boosting the performance.

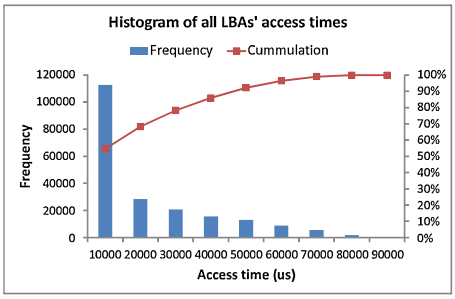
A previous work tried to solve this problem. In [3], ATCA is proposed to keep the blocks with large access times longer within the cache. By combining temporal locality and access time, this algorithm is found to achieve improved service time. But, we observe that such combination is not proper to deal with the high relationship between access frequency and access time of not a few requests, which can be often found in generic workloads reflecting real user activities. In this paper, we propose a novel disk cache algorithm to enhance the I/O performance by exploiting such relationship effectively, compared to ATCA.

2 Motivational observations

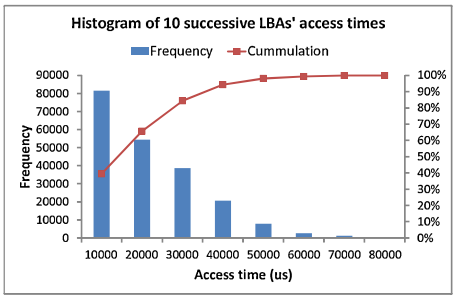
In order to find how access time is related with recency, we made some observations using a practical SATA disk simulator [3] with a cache off while running the PCFAT32 trace [4], which consists of storage accesses from real user activities of web surfing, word processing, presentation, and playing games, MP3 songs, and movies. The results can be seen in Fig. 1. Fig. 1(a) shows a histogram of all logical block addresses' (LBAs') access times and Fig. 1(b) shows a histogram of 10 successive LBAs' access times.

When we compare these two histograms, Fig. 1(a) shows that the proportion of small access times below 30000 us is 78.3 percent but most of them are found to be less than 10000 us while Fig. 1(b) shows that the proportion below 30000 us reaches 84.4 percent and access times are distributed more evenly. We notice that requests with temporal locality have a larger proportion of small access times than requests with no temporal locality. Thus, this observation indicates that such workload patterns sometimes may not give beneficial results for the ATCA algorithm since this algorithm tries to make requests with large access times as well as temporal locality stay longer by giving them high weights.

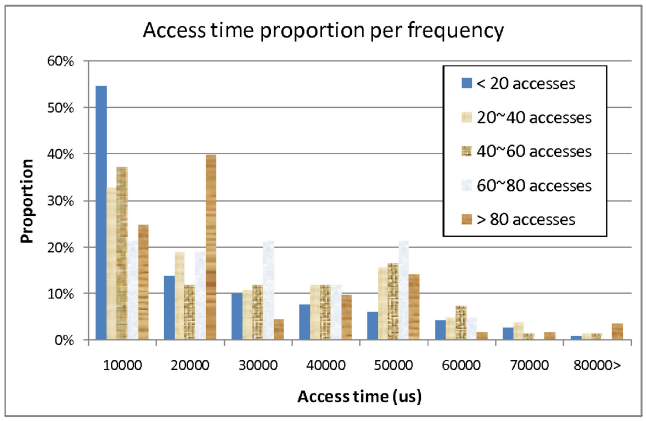
DOI: 10.1587/elex.11.20140363
Received April 17, 2014
Accepted April 21, 2014
Publicized May 14, 2014
Copyedited June 10, 2014



(a) all LBAs' access times



(b) 10 successive LBAs' access times



(c) LBA access time proportion per access frequency

Fig. 1. Comparisons of relations between recency/frequency and access time

Fig. 1(c) shows the observed relation between LBA access frequency and LBA access time. The graphs are shown to compare the access time proportion of each access frequency group. The LBA access frequency is categorized into 5 groups shown in Fig. 1(c): less than 20 accesses, 20∼40 accesses, 40∼60 accesses, 60∼80 accesses, more than 80 accesses. In the case of the first group, that is, less than 20 accesses, we notice that the interval of [0, 10000 us] shows the highest proportion of about 55 percent and the second highest proportion of about 14 percent can be found at the interval of (10000 us, 20000 us].

Fig. 1(c), the proportion of the access times below 30000 us amounts to 61.1 percent, 61.8 percent, and 69 percent for 40∼60, 60∼80, and more than 80 LBA access frequency, respectively. A noticeable thing is that the access time tends to become large as the frequency increases, compared to the result of temporal locality in Fig. 1. Thus, since keeping LBAs with large access frequency to stay within the cache can make not a few LBAs with large access times staying longer within the cache also, combining LBA access frequency and LBA access time will be very beneficial to caching valuable blocks for the purpose of enhancing I/O performance.

Based on our observations, we devise a novel disk cache management algorithm which gives a higher priority to a block with larger access time or higher access frequency within the cache. Our aim is to optimize the overall I/O performance by making requests with big access times or ones with high access frequencies stay longer in the cache. We believe that such two types of

DOI: 10.1587/elex.11.20140363
Received April 17, 2014
Accepted April 21, 2014
Publicized May 14, 2014
Copyedited June 10, 2014

requests may have more chances of overlap by our proposed algorithm than by LRU and this will be significantly beneficial to enhancing the I/O performance.

3 Proposed cache management algorithm

In order to devise a novel cache algorithm, we tackle modifying the least frequently used (LFU) cache algorithm by merging the concept of access time to the algorithm. To this end, a worth value is employed in a way how a normalized value of access time will be added to the access frequency update of the worth value. The process of achieving this normalized value of access time is shown in Fig. 2(a).

We need to normalize the value of access time first since it may be sometimes too big to be used together with other worth values in its current form. Basically we will classify the access time into the partitions numbered between 1 and a specific value such as 10. To classify the access time, we use the following formula:

$$N = Partition \times \frac{AccessTime}{MaxAccessTime},$$

where $Partition$ is the number of partitions and $AccessTime$ is the access time of each request, and $MaxAccessTime$ is the maximum possible access time. The $MaxAccessTime$ can be obtained statically or dynamically.

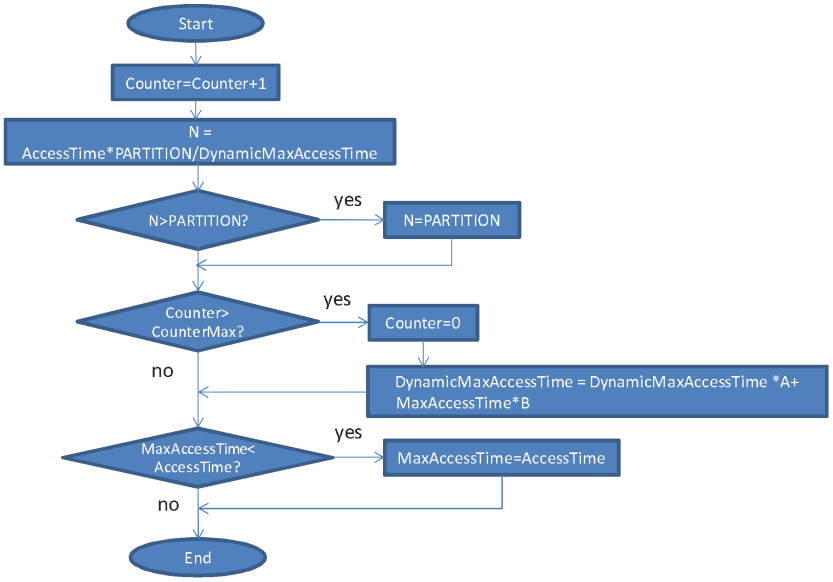
In the earlier stage of getting the $MaxAccessTime$, we considered a static policy. In this policy, we set this value to be the longest time possible for a disk head to reach a block. After some experiments, we found that the partition number of each request tends to be small and thus the request rarely get any large value for N. This affected the performance of the access time-aware caching algorithm since most of requests fell into the small partition values and thus the algorithm could hardly differentiate between big and small access times.

So, we decided to make the $MaxAccessTime$ vary dynamically. This means that we will record each request's access time, and keep track of the biggest access time value adaptively at runtime. With this approach, we can get more even partition values among all requests.

To further refine the mechanism, the maximum access time will be calculated based on a weighted average of the previous maximum access time and the current $MaxAccessTime$ values as shown in Fig. 2(a). This is to mitigate deviation of the N value due to the increase in the access time and thus to distribute the N value across the partitions more evenly.

In Fig. 2(b) and Fig. 2(c), we can see the examples where we run the process of getting the dynamic maximum access time. In both figures, the upper part shows the state before we run the process and the lower part shows the state after we finished the dynamic maximum access time process. In Fig. 2(c), *Dynamic Max Access Time* is updated since the counter reached 10, which is *CounterMax* in Fig. 2(a).

DOI: 10.1587/elex.11.20140363
Received April 17, 2014
Accepted April 21, 2014
Publicized May 14, 2014
Copyedited June 10, 2014



(a) Algorithm

Block=20
Access Time=200

Counter=5
DynamicAccessTime=1000
MaxAccessTime=2000

N=200*10*/1000=2

Counter=6
DynamicAccessTime=1000
MaxAccessTime=2000

(b) Example 1

Block=20
Access Time=2200

Counter=10
DynamicAccessTime=1500
MaxAccessTime=2000

N=2200*10*/1500=14.7
N=10

Counter=0
DynamicAccessTime=1500*0.2+2000*0.8=1900
MaxAccessTime=2200

(c) Example 2

Fig. 2. Algorithm and examples of normalizing the access time

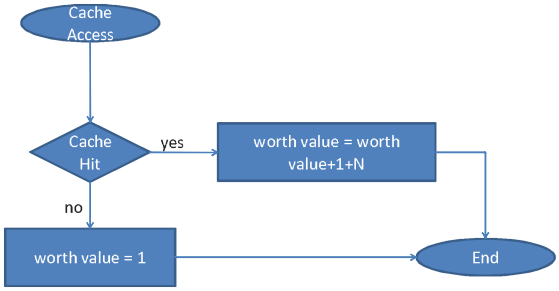


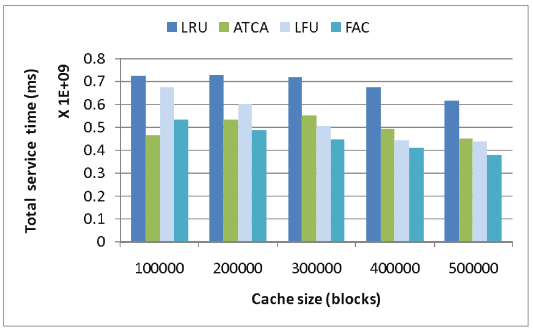
Fig. 3. Flow of the FCA algorithm

Fig. 3 shows the *Frequency-based Access time-aware Cache* (*FAC*) algorithm. It utilizes the value of a normalized access time, N, which comes from the previous stage of the algorithm shown in Fig. 2(a), to update the worth value of each block hit at the cache. This algorithm will make the data with bigger access times or frequent accesses stay longer in the cache, and make the data with smaller access times or infrequent accesses exit the cache soon.

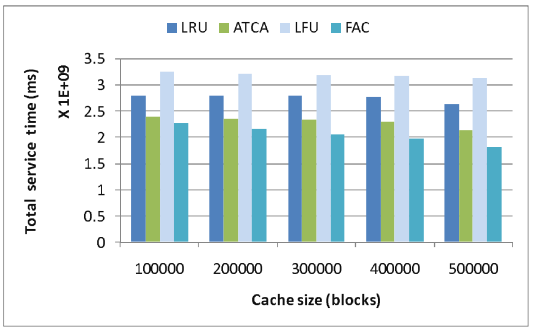
4 Experiments

To evaluate LRU, LFU, ATCA, and FAC, we used a practical SATA disk simulator [3], which simulates a realistic SATA disk model on real traces,

DOI: 10.1587/elex.11.20140363
Received April 17, 2014
Accepted April 21, 2014
Publicized May 14, 2014
Copyedited June 10, 2014



(a) For SMALLDISKMON



(b) For PCFAT32

Fig. 4. Total service times of LRU, ATCA, LFU, and FAC

consisting of a host controller and a disk controller. The host controller mimics the operating system of a host system and manages the requests transferred to the disk controller. The disk controller assumes all disk I/O operations including a cache, an NCQ, and a disk mechanic.

Two traces which gathered real generic user activities on a PC and a laptop were used for simulations: SMALLDISKMON trace [5] and PCFAT32 trace [4]. Their random requests reach 79.5 percent and 69.7 percent, respectively. And we employed the total service time as a metric. Total service time is the total time when all requests are served completely after they entered the queue.

Fig. 4(a) and 4(b) show the results of LRU, ATCA, LFU, and FAC for SMALLDISKMON and PCFAT32 traces. We notice that FAC shows the best performance among all evaluated cache algorithms in the aspect of the total service time. In detail, FAC achieved on average 35.1 percent and up to 39.2 percent improvement over LRU with the cache size varying for the SMALLDISKMON trace.

FAC also outperformed ATCA by 9 percent on average and up to 19.1 percent for the same trace although in case of the cache size of 10000 blocks, the total service time is increased by 14.5 percent once. We believe that such performance enhancement comes from that FAC fulfills effectively maintaining valuable requests and making them stay longer within the cache by giving large weights to the requests with high access frequency or large access time, which makes FAC more NCQ-friendly than ATCA. We also conjecture that a high proportion of random requests in the trace facilitates such a phenomenon more.

DOI: 10.1587/elex.11.20140363
Received April 17, 2014
Accepted April 21, 2014
Publicized May 14, 2014
Copyedited June 10, 2014

A similar result is shown for the PCFAT32 trace. With this trace, FAC is found to have on average 25.4 percent and up to 30.7 percent improvement over LRU. Compared to ATCA, FAC achieved on average 10.7 percent and up to 14.5 percent performance enhancement. NCQ gives random requests more chances to improve the I/O performance and since the FAC algorithm serves random requests more effectively than ATCA, its operation is more beneficial to enhance the performance of NCQ than ATCA and LRU.

5 Conclusion

In order to enhance the performance of SATA hard disk, we proposed a new NCQ-aware cache algorithm called FAC, which combines access time and access frequency to keep valuable cache blocks staying longer in the cache. For evaluation, we employed a trace-based SATA hard disk simulator with realistic models of a cache, an NCQ, and a disk mechanic at the level of a disk controller. Trace-driven simulations showed that our proposed algorithm achieves up to 39.2 and 19.1 percent performance improvement over LRU and ATCA, respectively, in the aspect of the total service time.

DOI: 10.1587/elex.11.20140363
Received April 17, 2014
Accepted April 21, 2014
Publicized May 14, 2014
Copyedited June 10, 2014