ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.9, No.21, 1707–1713

Access time-aware cache algorithm for SATA hard disks

Abdul Arfan[1], Young-Jin Kim[1a)], and Jin Baek Kwon[2]

[1] *Ajou University, San 5, Woncheon-dong, Yeongtong-gu, Suwon, 443–749, Republic of Korea*

[2] *SunMoon University, Kalsan-ri 100, Tangjeong-myeon, Asan, 336–708, Republic of Korea*

a) *youngkim@ajou.ac.kr*

Abstract: SATA has been widely used as the dominant hard disk storage technology. We tried to optimize the performance of SATA disks by devising a new cache algorithm that is aware of disk access time. To validate and test it, we implemented a simple but practical hard disk simulator. Real trace-driven simulations show that the proposed cache algorithm achieves up to 35 percent of improvement compared to LRU.

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

Classification: Storage technology

References

[1] Intel Corporation and Seagate Technology, "Serial ATA native command queuing: an exciting new performance feature for serial ATA," white paper, 2003.

[2] Y.-J. Kim, S.-J. Lee, K. Zhang, and J. Kim, "I/O performance optimization techniques for hybrid hard disk-based mobile consumer devices," *IEEE Trans. Consum. Electron.*, vol. 53, no. 4, pp. 1469–1476, Nov. 2007.

[3] D. M. Jacobson and J. Wilkes, "Disk scheduling algorithms based on rotational position," HP Technical report, 1991.

[4] L. Huang and T. Chiueh, "Implementation of a rotation latency sensitive disk scheduler," Technical Report ECSL-TR81, SUNY, Stony Brook, March 2000.

[5] L. Reuther and M. Pohlack, "Rotational-position-aware real-time disk scheduling using a dynamic active subset (DAS)," *Proc. 24th IEEE Int. Real-Time Systems Symposium*, Cancun, Mexico, pp. 374–385, Dec. 2003.

[6] J.-U. Kang, H. Jo, and J.-S. Kim, "A superblock-based flash translation layer for NAND flash memory," *Proc. EMSOFT'06*, pp. 161–170, 2006.

1 Introduction

The development of storage technology in computers has been making ceaseless progress and so many current computer applications need more storage

DOI: 10.1587/elex.9.1707
Received October 16, 2012
Accepted October 24, 2012
Published November 14, 2012

to store multimedia contents like movie and also large applications. Serial ATA (SATA) as the dominant hard disk storage technology advancement has replaced parallel ATA (PATA) in consumer desktop and laptop computers, and has largely replaced PATA in embedded applications. SATA's market share in the desktop PC market was 99% in 2008.

As a storage device, a hard disk has many mechanical components together with a platter and a head as well as many electronic components like a cache. With a cache, a hard disk can serve data better in terms of the performance. The Least Recently Used (LRU) algorithm has still been adopted by modern SATA hard disks as a cache algorithm, and we thus find little innovation in the area of cache algorithms. LRU is a well known cache algorithm, which discards the least recently used items first. But since this algorithm focuses on the recency of accessed data only, it has not employed components of a hard disk fully to enhance the performance.

We want to improve the performance of SATA disks by utilizing components in a disk well and thus extracting a novel controlling knob. To this end, we propose a new cache algorithm that is aware of access time of the disk. Since the disk knows the exact position of the disk head and which request will be accessed next, it can calculate the access time of each request. By utilizing this, we create a new cache algorithm that will keep a request that has a longer access time longer in the cache.

2 Behaviors of NCQ

One of SATA main features is native command queuing (NCQ). NCQ is a technology designed to increase performance of SATA hard disks under certain conditions. It optimizes the order in which received commands are executed. NCQ allows the hard disk to dynamically change the order in which the read/write requests are done.

From [1] we see that the best known algorithm to minimize both seek and rotational latencies is called Rotational Position Ordering. Rotational Position Ordering (or Sorting) allows the drive to select the order of command execution at the media in a manner that minimizes access time to maximize performance. Access time consists of seek time to position the actuator and latency time to wait for the data to rotate under the head. Earlier algorithms simply minimized seek distance to minimize seek time. However, a short seek may result in a longer overall access time if the target location requires a significant rotational latency period to wait for the location to rotate under the head. Rotational Position Ordering considers the rotational position of the disk as well as the seek distance when considering the order to execute commands. Commands are executed in an order that results in the shortest overall access time (i.e., the combined seek and rotational latency time) to enhance performance. NCQ allows a drive to take advantage of Rotational Position Ordering to optimally re-order commands to maximize performance.

Although NCQ is well designed to optimize the overall access time, it does not achieve much benefit for some hard disk usage such as loading a large

DOI: 10.1587/elex.9.1707
Received October 16, 2012
Accepted October 24, 2012
Published November 14, 2012

application. When an application is installed on a hard disk, all its contents are located around the same area of the disk. So, when an application is loaded the amount of hard disk head movement is naturally small. Moreover, since most applications are loaded sequentially, NCQ cannot make a big difference in disk access times.

NCQ will start to shine if we run two different applications simultaneously. Two applications competing for access at two different locations on a hard disk will benefit from NCQ. With the increase of multi threaded applications and the use of Hyper Threading technology, NCQ will gain the benefit.

A cache has been widely used to enhance the performance of a disk. However, the existing cache algorithm that is used along with NCQ such as LRU is not aware of access time. LRU only focuses on the temporal locality. This makes the cache algorithm not differentiate between a request with a big access time and one with a small access time. Thus, we try to devise an access time-aware cache algorithm to make full use of NCQ. Our algorithm will make requests with large access times stay longer in the cache. And it will evict requests with small access times out of the cache soon because they can be served by the disk sequentially without much movement of the head.

We tested our idea by devising a simple version of our cache algorithm and running it with our simulator. We realized a cache algorithm that will simply classify the access time of a request into a big access time or small one. For a big access time, we assigned a bigger value to the corresponding request so that it can stay longer in the cache due to a value-based eviction policy.

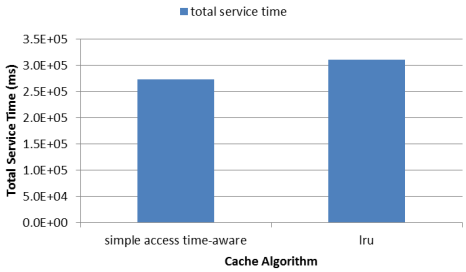


Fig. 1. Improvement from a simple access time-aware cache algorithm

We made an experiment with the algorithm by using a manually created trace file and simulating all requests within it. In Figure 1, we can see our proposed cache algorithm enhances the total service time by 11% over LRU.

3 Proposed cache algorithm

Based on the result of the motivational example, we propose a new cache algorithm for cache management in a hard disk, which is called the access time-aware cache algorithm (ATCA). The intuition for this algorithm is that we want to keep the blocks with large access times longer in the cache, thus giving them more chance to be accessed via a cache instead of a disk.

DOI: 10.1587/elex.9.1707
Received October 16, 2012
Accepted October 24, 2012
Published November 14, 2012

Each block in the cache will be assigned a value called worth value. This value will determine how worthy the block is in the cache. The block with a bigger worth value will presumably stay longer in the cache compared to one with a small worth value. In the LRU algorithm, if there is no space in the cache and we insert a new block to the cache, then we have to evict a block from the cache. In this case, we have to evict the least recently used block. In the ATCA algorithm, we will evict a block based on the worth value and the block with the smallest worth value will be evicted from the cache.

Figure 2 (a) shows the pseudo-code of the ATCA algorithm. This algorithm can be called a generalized LRU. LRU only considers the temporal locality of blocks. A block that is least accessed will be considered least important and thus it will be chosen to be evicted if the cache is full. In the ATCA algorithm, not only the temporal locality will be considered, but also the access time. We want to make the block that has a large access time stay longer in the cache.

To implement the ATCA algorithm, first we need to recreate the LRU algorithm in the worth value perspective, and then modify the algorithm to make it aware of the access time. The algorithm in Figure 2 (a) can be viewed as the LRU algorithm if N is zero. However, in the ATCA algorithm, N is not zero since it is the value of the access time. The M variable will keep the temporal feature in ATCA. M is used to reflect the temporal locality of a block. We combine M and N to make the algorithm have the temporal feature as well as awareness of the access time.

4 SATA hard disk simulator

To test our algorithm, we developed a SATA disk simulator. Figure 2 (b) shows our simulator architecture, which was inspired by [2] for the overall architecture and [3, 4, 5] to model components and scheduling algorithms. The simulator consists of 2 parts: host controller and disk controller. The former manages the requests which will be transferred to the disk controller. It mimics the operating system of a host system. The latter is responsible for all disk I/O operations. It consists of 3 parts: cache, queue, and disk mechanic.

The cache is used to store frequently-accessed data to speed up the disk I/O operation. We implemented LRU and ATCA to manage the cache. A queue is located behind a cache. It is the place where some requests are waiting to be served. Since the disk controller can serve multiple requests from the host, each request should wait in the queue, where the disk controller can do some useful operations such as changing the order of serving requests and also I/O clustering.

If there is still a space in the queue, then the simulator moves the request to the queue. Before the request in the buffer reaches the queue, its hit or miss will be checked at the cache. In the queue, multiple sequential requests can be merged into a large single one by I/O clustering. Merging will only occur if the total size of requests is smaller than the threshold (we call this an

DOI: 10.1587/elex.9.1707
Received October 16, 2012
Accepted October 24, 2012
Published November 14, 2012

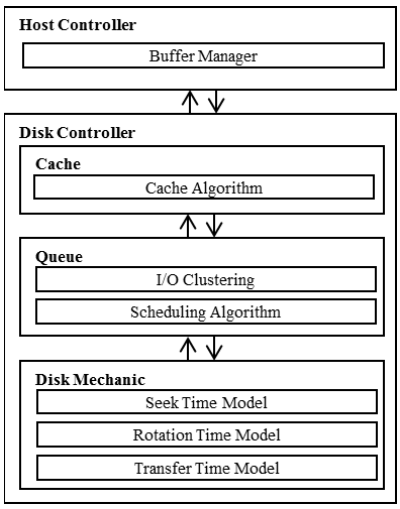
VARIABLES

X: accessed block
M: counter value
N: scaled value for the access time of X
$W(X)$: total worth value of X

MAIN ALGORITHM

1: $M \leftarrow M + 1$
2: If a cache hit occurs then
3: $\quad W(X) \leftarrow M + N$
4: Else
5: $\quad$ If there is no space in the cache then
6: $\quad\quad$ Evict a block with the least worth value
7: $\quad$ Add X to the cache
8: $\quad W(X) \leftarrow M$

(a)



(b)

Fig. 2. (a) Pseudo-code of the ATCA Algorithm (b) Architecture of the developed SATA disk simulator

I/O clustering threshold). This is to prevent arbitrary requests from being merged indefinitely. In the experiment, we set the default threshold to 32 LBAs.

SATA uses NCQ as its disk scheduler. In order to simulate a SATA disk, we thus need to implement the NCQ algorithm in our simulator. Since there is no detailed information on how NCQ is implemented in a real SATA disk, our implementation was done based on the basic behaviors of NCQ, which were described previously. We implement NCQ by choosing the next request that have the smallest seek time and rotational time from current head position. This algorithm is similar to Shortest Access Time First (SATF). And we also add a timeout policy to the scheduler. If the request is waiting too long in the queue, then it will be promoted to be served by the disk.

5 Simulation results

To evaluate a SATA disk using our simulator, we employ the total service time as a main metric. Total service time is the total time required by all requests so that they are served completely after they entered the queue. It can give a good profile of the overall performance of a hard disk.

Table I shows the information of two trace files that we used. They include arrival time, logical block address (LBA), and size of the request. SMALLDISKMON trace was created using *Diskmon* while running Internet Explorer, NATE-on (messenger program), MSN (MS messenger program), Microsoft PowerPoint, and Adobe Photoshop [2]. PCFAT32 was extracted from real user activities on the notebook of personal usage [6].

Figure 3 shows the result of improvement of the ATCA algorithm over LRU. We notice that ATCA can get up to 35 percent of performance improvement over LRU. ATCA will keep the requests with the big access times longer inside the cache, thus making them to be served fast from the cache.

DOI: 10.1587/elex.9.1707
Received October 16, 2012
Accepted October 24, 2012
Published November 14, 2012

Table I. Trace file information [2, 6].

Properties	SMALLDISKMON	PCFAT32
Working set (req./block)	41819/1956272	204386/7484939
Random (req./block)	57166(78.5%)/1872624(51.6%)	437742(68.3%)/10417692(41.7%)
Sequential (req./block)	15656(21.5%)/1756502(48.4%)	199145(31.3%)/14561873(58.3%)
Read (req./block)	31208/992172	414498/14724779
Write (req./block)	41614/2636954	222389/10254786

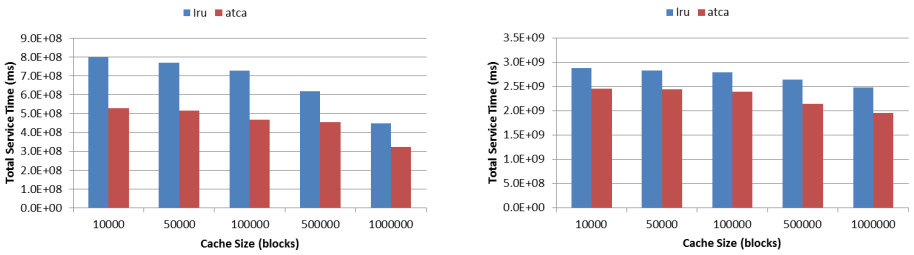
This approach will minimize the total access time and result in the minimized total service time.

Figure 3 (a) and (b) show the results of ATCA with SMALLDISKMON and PCFAT32 traces. ATCA achieves up to 35 percent of improvement in the total service time for the SMALLDISKMON trace and 21 percent for the PCFAT32 trace compared to LRU. Since LRU is not aware of access time, it cannot differentiate effectively which requests actually should stay longer in the cache. These results show that ATCA has some potential in improving the overall SATA disk performance.

ATCA can gain more improvement with the SMALLDISKMON trace than with the PCFAT32 trace because there are more sequential requests in the PCFAT32 trace. In Table I, we can see sequential requests reach 21.5 percent among all in the SMALLDISKMON trace compared to 31.3 percent in the PCFAT32 trace. In the aspect of blocks, the sequentiality goes further. Sequential requests do not need much head movement compared to random (i.e., non-sequential) requests. NCQ will have better effect on non-sequential requests. Since the ATCA algorithm tries to serve non-sequential requests more from the cache and serve sequential requests from the disk, its operation is beneficial to enhance the performance of NCQ effectively.

6 Conclusion

In order to enhance the performance of SATA hard disk, we developed a new cache algorithm called ATCA, which considers access time as well as temporal locality in managing cache blocks. For evaluations, we developed a practical SATA hard disk simulator. We added realistic models for components including a cache, a queue, and a disk mechanic at the level of a disk controller and tried to validate their operations. Trace-driven simulations



(a) (b)

Fig. 3. Total service times of LRU and ATCA
(a) for SMALLDISKMON (b) for PCFAT32

DOI: 10.1587/elex.9.1707
Received October 16, 2012
Accepted October 24, 2012
Published November 14, 2012

showed that our proposed algorithm achieves up to 35 percent performance improvement over LRU in the aspect of the total service time.

Acknowledgments

This work was supported by the new faculty research fund of Ajou University. This research also was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (no. 2011-0005386).

DOI: 10.1587/elex.9.1707
Received October 16, 2012
Accepted October 24, 2012
Published November 14, 2012