ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.12, No.4, 1–6

An efficient scheduling algorithm for NCQ within SSDs

Yongwoon Cho and Taeseok Kim[a)]

Department of Computer Engineering, Kwangwoon University,
447–1, Wolgye-Dong, Nowon-Gu, Seoul, Korea

a) *tskim@kw.ac.kr*

Abstract: An efficient scheduling scheme for NCQ in SSDs is presented. Our scheme estimates the I/O service time of each command by considering the state of the buffer as well as the SSDs' physical characteristics, and then first services the command with the shortest I/O service time. Through the trace-driven simulations, we show that the proposed scheme significantly improves the I/O performance in terms of average response time.

Keywords: Solid State Drives (SSDs), NCQ (Native Command Queueing), scheduling

Classification: Storage technology

References

[1] A. M. Caulfield, L. M. Grupp and S. Swanson: Proc. of the 14th ASPLOS (2009) 217. DOI:10.1145/1508244.1508270

[2] J. Kim, Y. Oh, E. Kim, J. Choi, D. Lee and S. H. Noh: Proc. of the ACM International Conference on Embedded Software (2009) 295. DOI:10.1145/1629335.1629375

[3] Y. Kim and T. Kim: Electron. Lett. **47** (2011) 482. DOI:10.1049/el.2010.3593

[4] Y. J. Yu, D. I. Shin, H. Eom and H. Y. Yeom: ACM Trans. Storage **6** (2010). DOI:10.1145/1714454.1714456

[5] G. Gasior: Intel's X25-E Extreme Solid-state Drive, *The Technical Report* (2008).

[6] S. Kim and T. Kim: Electron. Lett. **49** (2013) 1079. DOI:10.1049/el.2013.1637

[7] OpenSSD project, http://www.openssd-project.org/wiki/The_OpenSSD_Project.

[8] A. Silberschatz, P. B. Galvin and G. Gagne: *Operating System Concepts* (John Wiley & Sons, 2012) 9th ed.

[9] D. D. Levine: Iometer User's Guide, *Intel Server Architecture Lab*. (2003).

1 Introduction

With the fast advance in storage technology, hard disk drives are being rapidly replaced with NAND flash memory based SSDs (Solid State Drives) in most computing environments. Unlike hard disk drives, SSDs have little data access overhead due to the absence of disk head, asymmetric read and write operation speed, and limited write endurance [1]. For this reason, the existing various storage

© IEICE 2015
DOI: 10.1587/elex.12.20150066
Received January 20, 2015
Accepted January 21, 2015
Publicized February 3, 2015
Copyedited February 25, 2015

system software techniques designed for hard disk drives are being redesigned by considering the characteristics of SSDs [2, 3]. In this paper, we focus on NCQ (Native Command Queueing), which is one of the techniques that improve the I/O performance of hard disk drives [4].

NCQ is an extension of the Serial ATA protocol that allows hard disk drives to internally optimize the order of commands, and it is still widely employed in SSDs [5]. In hard disk drives, it was efficient to service commands in NCQ by minimizing the number of disk head rotations, but it may be not efficient in SSDs if the scheme is employed without any modification because SSDs don't have disk head. Based on this fact, we propose an efficient scheduling scheme for NCQ in SSDs by considering the characteristics of SSDs: little data access overhead and asymmetric read and write speed. In addition to this, our scheme maximizes the I/O performance by considering the state of the write buffer, which exists in most SSDs in order to improve the I/O performance and lifetime.

Since there is little data access overhead in SSDs, it is not easy to improve the IOPS or I/O bandwidth, but it is possible to optimize the average response time. Our scheme reduces the average response time by estimating the I/O service times of all commands in NCQ and then first servicing the command with the shortest I/O service time. Through the trace-driven simulations, we show that the proposed scheme improves the I/O performance in terms of the average response time without any loss of IOPS and I/O bandwidth.

2 NCQ scheduling scheme considering the SSD's internals

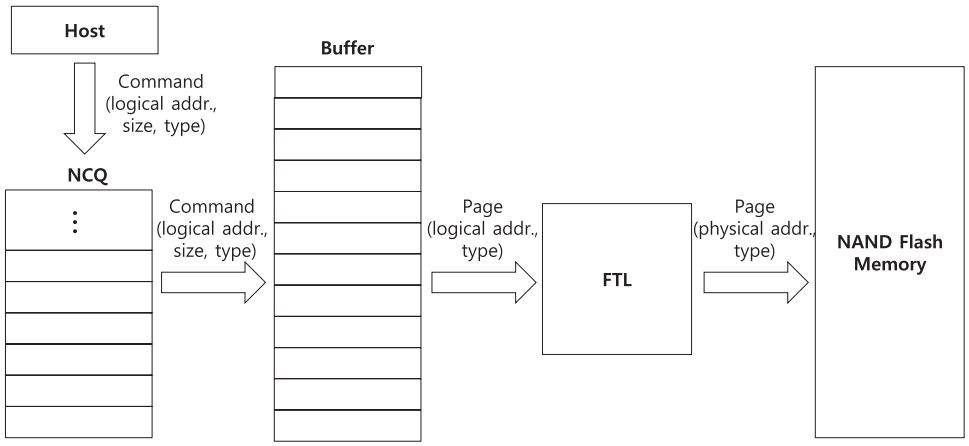


Fig. 1. The procedure of an I/O command handling in SATA SSDs.

To understand the proposed NCQ scheduling scheme, it is necessary to first carefully observe the internal architecture of SSDs with SATA interface. Fig. 1 illustrates the procedure that handles a command in SSDs [6, 7]. The command arriving from the host consists of logical address, size, and type, and it is queued in NCQ. Once servicing a previous command is completed, another command at the head of NCQ is dispatched. If data belonging to the dispatched command resides in buffer, it can be served from the buffer; otherwise it is finally serviced from NAND flash memories via the FTL (Flash Translation Layer).

© IEICE 2015
DOI: 10.1587/elex.12.20150066
Received January 20, 2015
Accepted January 21, 2015
Publicized February 3, 2015
Copyedited February 25, 2015

The proposed scheme estimates the I/O service time of each command in NCQ, and then first services the command with the shortest I/O service time. It is analogous to the idea of the Shortest Job First scheme, which minimizes the average response time by first servicing the process with the shortest CPU burst time [8]. The I/O service time in hard disks is typically modeled as the sum of seek time, rotational delay, and data transfer time. However, SSDs have no seek time and rotational delay, so the I/O service time in SSDs depends only on the data transfer time. In turn, the data transfer time is proportional to the size of command. Since the write operation is much slower than the read operation in SSDs, the type should also be considered for estimating the I/O service time. Finally, we consider if data belonging to the command resides in the buffer or not. This is because if data belonging to a command resides in buffer, it can be served from the buffer; otherwise, it is required to read or write NAND flash memory, and thus it makes a significant effect on I/O service time.

In order to consider the size, type, and buffer state all together for NCQ scheduling, we evaluate the I/O service time of each command as follows. First, since the I/O service time is proportional to the size of command, we can model the I/O service time of the ith command, $T_{io}(i)$ as (1) if we consider only the size.

$$T_{io}(i) = N(i) \times T_{page} \tag{1}$$

In (1), $N(i)$ is the number of pages to be read or written for the ith command and T_{page} is the time taken for reading or writing a page. As the read and write operation speeds are different, if the type is also considered, (1) can be classified as (2) according to the type.

$$\begin{aligned} T_{io}(i) &= N(i) \times T_{read}, \quad \text{for read} \\ T_{io}(i) &= N(i) \times T_{write}, \quad \text{for write} \end{aligned} \tag{2}$$

In (2), T_{read} is the time taken for reading a page from NAND flash memory and T_{write} is the time for writing a page to NAND flash memory, respectively.

If data belonging to a command resides in the buffer, it can be serviced from the buffer instead of NAND flash memory. Generally, SDRAM is used for buffer, and the access time to SDRAM can be neglected because it is much smaller than the access time to NAND flash memory. In conclusion, we can model the I/O service time as the time taken for servicing only the pages missing in buffer. If we additionally consider the buffer state, (3) can be derived. In (3), $N_{miss}(i)$ is the number of pages missed in buffer for the ith command.

$$\begin{aligned} T_{io}(i) &= N_{miss}(i) \times T_{read}, \quad \text{for read} \\ T_{io}(i) &= N_{miss}(i) \times T_{write}, \quad \text{for write} \end{aligned} \tag{3}$$

For write commands, we need to describe the above expression more exactly. In case of page hit, it is sufficient to update the pages in buffer without any flush operation, but the missed pages should be newly written to buffer. If there is no empty buffer, as many pages as $N_{miss}(i)$ should be first replaced from buffer in order to provide empty space for the missed pages. It is an ordinary situation because there is usually no empty buffer due to the frequent I/O operations. For buffer replacement, we employ a widely used LRU (Least Recently Used) strategy that first replaces the least recently used buffer [8].

© IEICE 2015
DOI: 10.1587/elex.12.20150066
Received January 20, 2015
Accepted January 21, 2015
Publicized February 3, 2015
Copyedited February 25, 2015

Since the proposed scheme gives higher priority to the commands with short I/O service time in order to reduce average response time, the commands with long I/O service time may experience starvation. To alleviate this problem, we supplemented our model with a simple aging scheme. In other words, whenever a command in NCQ is overtaken by other newly-arrived commands, its I/O service time can be revised as in (4).

$$T_{io}(i) = \alpha^{m_i} \times N_{miss}(i) \times T_{read}, \quad \text{for read} \qquad (4)$$
$$T_{io}(i) = \alpha^{m_i} \times N_{miss}(i) \times T_{write}, \quad \text{for write}$$

In (4), α is a weight value for aging, and is between 0 and 1. m_i is the number of commands that overtake the ith command while it is waiting in NCQ. Since α is less than 1, whenever a command in NCQ is overtaken by the newly arrived other commands, the I/O service time decreases, and thus the starvation will not occur. In conclusion, when α is close to 0, the proposed scheme will behave like FCFS (First Come First Served); when α is close to 1, it will be close to (3).

3 Performance evaluations

To demonstrate the effectiveness of the proposed scheme, we implemented a simulator that emulates the I/O handling in SSDs with SATA interface. Table I lists SSD parameters used in our experiments. We assume that the SSD has several flash packages that consist of chips [1]. As each chip can operate independently, I/O operations are performed in parallel with a clustered unit consisting of several pages from different chips.

Table I. SSD parameters configured.

SSD capacity	64 GB
Packages/SSD	8
Chips/package	8
Planes/chip	2
Blocks/plane	2048
Pages/block	64
Page size	4 KB
Page read latency	0.02 ms
Page write latency	0.2 ms
Block erase latency	1.5 ms
Queue depth	128

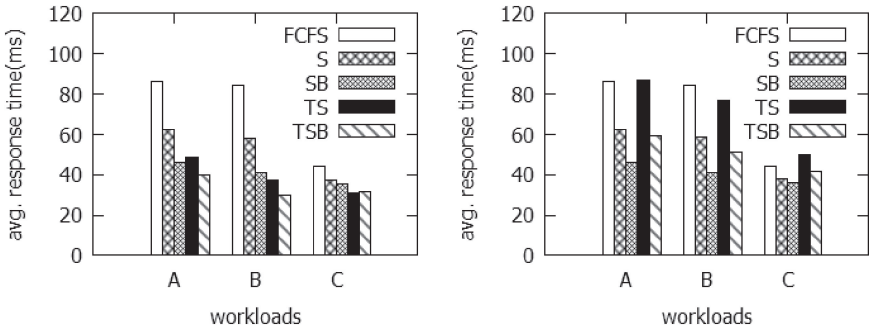
In order to extensively analyze the effect of each consideration: the I/O size, the I/O type, and the buffer state, we compared several intermediary algorithms. S considers only the I/O size, SB considers both the I/O size and buffer state, TS considers both the I/O type and I/O size, and TSB considers them all, which is the proposed algorithm. For evaluating our scheme in different environments, we gathered various workloads from IOmeter [9], which are listed in Table II.

© IEICE 2015
DOI: 10.1587/elex.12.20150066
Received January 20, 2015
Accepted January 21, 2015
Publicized February 3, 2015
Copyedited February 25, 2015

Table II. Characteristics of the workloads used.

	Workload A	Workload B	Workload C
Number of threads	12	12	12
File size	512 KB–1 GB	512 KB–1 GB	512 KB–1 GB
Record size	4 KB–512 KB	4 KB–512 KB	4 KB–512 KB
Avg. inter-arrival time	1.78	1.76	1.08
Read/write ratio	2:1	1:1	2:1
Access pattern	random	random	sequential

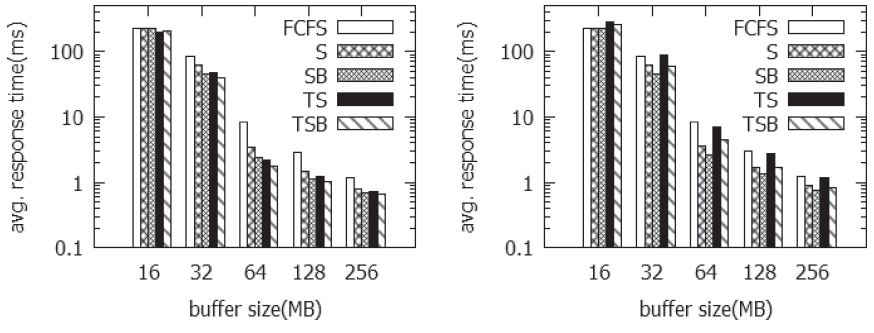
Fig. 2 shows the average response time of several algorithms for different workloads. In this experiment, α is 0.9, and buffer size is 32 MB. As expected, in all workloads used, the average response time is significantly reduced when I/O size and buffer state are considered together. If the I/O type is additionally considered, the average response time of read commands decreases more, but that of write commands increases. Compared to FCFS, our scheme that considers them all reduces the average response time of read commands up to 64%, 75%, and 29% with workload A, B, and C, respectively.



(a) Avg. read response time (b) Avg. write response time

Fig. 2. Response time with different workloads.

Fig. 3 shows the average response time as a function of buffer size when workload A is used. Irrespective of buffer size, our scheme shows better performance than FCFS. In particular, when the buffer size is 32 MB, the average response time of read commands in FCFS is 86 ms; it is reduced to 62 ms when I/O size is considered; it is 46 ms when both I/O size and buffer state are considered together;



(a) Avg. read response time (b) Avg. write response time

Fig. 3. Response time as a function of buffer size.

© IEICE 2015
DOI: 10.1587/elex.12.20150066
Received January 20, 2015
Accepted January 21, 2015
Publicized February 3, 2015
Copyedited February 25, 2015

and it becomes 39 ms when all considerations are exploited. In this experiment, the IOPSs of FCFS and TSB are 636 and 638, and the I/O bandwidths of FCFS and TSB are 65,811 KB/s and 65,966 KB/s, respectively.

4 Conclusions

In this paper, we presented a novel scheme for NCQ scheduling. Our scheme first estimates the I/O service time of each command by considering the I/O size, I/O type, and buffer state, and then first services the command with the shortest I/O service time. Through extensive experiments, we demonstrated that our NCQ scheduling is very effective in terms of average response time.

Acknowledgments

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012-0001924). This research has been also conducted by the Research Grant of Kwangwoon University in 2013.

© IEICE 2015
DOI: 10.1587/elex.12.20150066
Received January 20, 2015
Accepted January 21, 2015
Publicized February 3, 2015
Copyedited February 25, 2015