Performance Enhancement of I/O Scheduler for Solid State Devices

Seungyup Kang, Hyunchan Park, Chuck Yoo
Department of Computer Science and Engineering, Korea University, South Korea

***Abstract*-- Due to the benefits of Solid State Drive (SSD), it has been called a *pivotal technology* on data storage systems. But, current device level I/O schedulers are not optimized for SSD. In this paper, we suggest a new I/O scheduler, called STB, to exploit the performance potential of SSDs. Our STB scheduler categorizes I/O requests into two groups and sets timers on each request. We implement STB scheduler in Linux 2.6.30, and the set of benchmark programs shows that STB scheduler provides better bandwidth up to 30% with low response time of each request.**

I. INTRODUCTION

In recent years, SSD has become an emerging storage device in both industry and academia[1][2]. SSD is widely embedded in mobile consumer electronics such as portable media player and lap-top computer, because of its benefits such as low energy consumption, shock resistance and compact size. It is also adopted in desktop computers and enterprise servers because it has uniform and fast access time. In those areas, SSD can provide very high bandwidth compared to HDD. For the reasons, it is considered as a *pivotal technology* to a new generation of storage devices.

During the last years, researchers have made continuous efforts to exploit the performance of SSDs. Some researchers have attempted to revise whole existing I/O subsystems[3]. However, they cannot guarantee backward compatibility. In order to maximize performance of SSD with backward compatibility, I/O schedulers have been focused on [4][5]. Dunn and Reddy[4] have presented a framework for extracting device parameters of SSD; however, their results do not overcome the performance of traditional Linux I/O scheduler. Kim et al.[5], the latest technology, show that appropriate bundling of write requests into Logical Block Allocation size eliminates any ordering related restrictions so that it can increase write performance; but, they hastily group write requests without any concern that they lose the opportunity to improve performance throughput further.

In this paper, we propose a new I/O scheduler, called Selective Timeout Bundling (STB, hereafter), for SSD to exploit the performance potential of the SSDs. For this purpose, we propose two policies; 1) categorize requests into selective groups, and 2) set timers for each request to prevent request from write starvation. We implement our STB scheduler in Linux 2.6. 30. As a result, STB scheduler shows performance throughput improvements of up to 30% and low response time compared with existing Linux I/O schedulers.

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2010-0029180, No. 2010-0016637) and by the Seoul Research and Business Development Program, Smart(Ubiquitous) City Consortium and Seoul Grid Center (No. 10561, No. WR080951).

II. BACKGROUND

A. Traditional Linux I/O Schedulers

The most recent Linux kernel 2.6 provides four I/O schedulers; *No-operation (Noop), Deadline, Anticipatory, and Complete Fair Queueing (CFQ) Scheduler*. Those schedulers are based on Elevator algorithm that is invented for HDD. The algorithm sorts and merges I/O requests to reduce movement of disk head, but response time of the request can increase due to CPU processing. These I/O schedulers are not effective on SSD because it does not have disk heads unlike HDD. For these reasons, we need a new I/O Scheduler based on characteristics of SSD.

B. SSD Features

The SSD is composed of NAND flash memory modules by connecting them in parallel. To support the main feature of NAND flash module such as wear-leveling and out-place update, the SSD has an own embedded system, Flash Translation Layer (FTL, hereafter). FTL maps logical blocks to physical blocks. The goal of mapping algorithm is to fully utilize the consisted flash modules by maximizing parallelism[1]. For the purpose, FTL defines Logical Block (LB, hereafter) for mapping. The size of LB is usually defined more than 1MB because the size of physical blocks are typically varying from 2KB to 128KB and the number of flash modules integrated into a SSD is typically 4 to 10. FTL also performs a garbage collection which issues many reads and writes internally.

C. IRBW-RP Scheduler

Based on above features, Kim et al. propose *Immediately Read and Bundling Write-Read Preferences* (IRBW-RP, hereafter) scheduler for SSD. The main feature of IRBW-RP is bundling write requests to maximize bandwidth. The size of one bundle is same with the size of LB. It can improve bandwidth without a decrement of response time compared with traditional I/O schedulers.

In spite of the improvement on bandwidth, we found that there is another potentiality for further advancement. Although they remove the sorting and merging processes from traditional schedulers, any improvement on response time is not observed. Moreover, they do not distinguish the type of write requests for bundling; there may be synchronous write requests such as a request for file system m etadata and a request with O_DIRECT flag from user applications. Because those requests can block the file system or the application, overall performance can be degraded when many I/O-

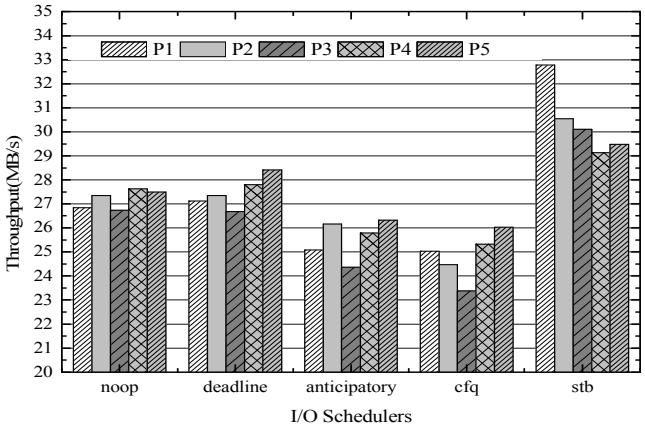

Fig. 1. Results of Postmark Benchmark (Higher is better). Y-axis is the throughput in MB/sec. X-axis is the type of I/O scheduler

intensive processes are running simultaneously in enterprise server environments.

III. STB: NEW I/O SCHEDULER

We are inspired by the potentialities, we propose a new I/O scheduler for SSD. In this section, we describe the design of our scheduler, named STB. Based on design of IRBW-RP scheduler, we suggest two additional features.

A. Time-Out-Bundling

Since SSDs use block mapping FTL which regards multiple blocks as a logical block, the logical block size will be a key factor to determine write throughput. Moreover, in order to provide reasonable response time for each request, we invent the time-out-scheme which pauses requests a short period not only because of bundling efficiently, but also because of guaranteeing convincible response time for each request. By adopting time-out-scheme to each request, we can prevent requests from waiting other requests to reach LB size.

B. Selective Bundling

Due to the types of requests are various, we categorize requests into two groups by its innate feature; synchronous requests or asynchronous requests. Selection of requests is very important because some requests such as read and write on file system metadata block possibly block whole requests of other processes. So, in order to prevent worst case, we have to deal synchronous requests as soon as possible.

We can explain dispatching policy by three main routes. First, we dispatch a bundle of requests if it reaches a LB size. Second, we dispatch a request of synchronous group immediately since these requests possibly block other requests. So, we prioritize synchronous requests. Finally, we give each request check-point-time. This check-point-time guarantees that each request have convincible deadline that prevents from being continuously waiting for additional requests to be reach LB size.

IV. EXPERIMENTAL RESULTS

We implement the new scheduler design in the Linux 2.6.30 kernel. We evaluate our STB scheduler and compared with traditional Linux I/O schedulers using the Postmark benchmark program version 1.5. We execute the five Postmark processes concurrently to evaluate a realistic performance. Moreover, much more I/O requests will reduce the buffer cache effect in experiments. It leverages that we can evaluate I/O schedulers more precisely.

TABLE I
AVERAGE AND WORST CASE RESPONSE TIME OF OPERATIONS

Average.	read()	write()	open()	create()
Noop	211.324	12.28	33.61	850.892
Deadline	248.15	11.608	38.442	1344.042
STB	**158.986**	11.716	42.364	**634.754**
Worst case	read	write	open	create
Noop	133703	1944140	66224	235195
Deadline	98966	345291	90511	1153734
STB	**94774**	403423	80121	**165415**

* Results are shown in nano seconds.

The results of Postmark benchmarks are presented in Fig. 1. In Fig. 1, we can see that STB scheduler achieves better throughputs than the others. Table 1 shows the average and maximum response time of each various operations; read, write, open and create operation. Due to open and write operations are processed without I/O request, very low response time was shown. Because a read request is dispatched faster than others, response time of read on STB is lower than others. Response time of create time is important factor on realistic environment because of the blocking caused by modification of file system metadata. We conclude that the higher overall bandwidth of STB is belongs from the fast response time of creation.

V. CONCLUSION

In this paper, we suggest that new I/O scheduler for SSD which has better throughput than other existing Linux I/O schedulers. In order to attain performance benefits, we propose time-out-bundling and selective bundling schemes. Time-out-bundling scheme prevents requests from waiting other requests infinitely. At the same time, selective bundling mechanism give way to discriminate performance critical requests, so that it prevents from being blocked. The results shows that our STB scheduler improves performance up to 30% than other Linux I/O schedulers.

REFERENCE

[1] Agrawal, N., V. Prabhakaran, et al. “Design tradeoffs for SSD performance”, *USENIX Association*, pp. 57-70, 2008.

[2] Chen, F., D. Koufaty, et al. "Understanding intrinsic characteristics and system implications of flash memory based solid state drives", *ACM SIGMETRICS*, pp. 181-192, 2009.

[3] Soundararajan, G., V. Prabhakaran, et al., "Extending SSD Lifetimes with Disk-Based Write Caches"*, USENIX FAST*, 2010.

[4] Dunn, M. and A. Reddy. "A new I/O scheduler for solid state devices." *Texas A&M University ECE Technical Report TAMU-ECE-2009-02*, 2009.

[5] Kim, J., Y. Oh, et al. "Disk schedulers for solid state drivers", proc. of the 7th *ACM international conf. on Embedded software*, pp. 295-304, 2009.