On the Parallelism of I/O Scheduling Algorithms in MEMS-Based Large Storage Systems

EUNJI LEE and KERN KOH
School of Computer Science and Engineering
Seoul National University
56-1 Shillim-dong, Kwanak-gu, Seoul, 151-742
REPUBLIC OF KOREA
{ejlee, kernkoh}@oslab.snu.ac.kr http://oslab.snu.ac.kr

HYUNKYOUNG CHOI and HYOKYUNG BAHN*
Department of Computer Science and Engineering
Ewha University
11-1 Daehyun-dong, Seodaemun-gu, Seoul, 120-750
REPUBLIC OF KOREA
bluechk@nate.com; bahn@ewha.ac.kr http://home.ewha.ac.kr/~bahn

Abstract: MEMS-based storage is being developed as a new storage media that has several salient characteristics such as high-parallelism, high density, and low-power consumption. Because physical structures of MEMS-based storage is different from those of hard disks, new software management techniques for MEMS-based storage are needed. Specifically, MEMS-based storage has thousands of parallel-activating heads, which requires parallelism-aware request scheduling algorithms to maximize the performance of the storage media. In this paper, we compare various versions of I/O scheduling algorithms that exploit high-parallelism of MEMS-based storage devices. Trace-driven simulations show that parallelism-aware algorithms can be effectively used for high capacity mass storage servers because they perform better than other algorithms in terms of the average response time when the workload intensity becomes heavy.

Key-Words: MEMS-based storage, Parallelism, Request Scheduling, Scheduling algorithm, Storage.

1 Introduction

MEMS-based storage is one of the leading candidates as tomorrow's storage medium. Due to its salient characteristics such as high-parallelism, low-power consumption, low cost, and high-density, MEMS-based storage is anticipated to be used for a wide range of applications from storage for small handheld devices to high capacity mass storage servers [11-14]. However, MEMS-based storage has a couple of different physical characteristics compared to a traditional hard disk [1-4]. First, MEMS-based storage has thousands of heads that can be activated simultaneously. Second, the media of MEMS-based storage is a square structure, which is different from the rotation-based platter structure of disks. Due to these differences in characteristics, new system software technologies appropriate for this media have been issues of recent research [2, 3].

Similar to hard disks, one of the most important management mechanisms for improving MEMS-based storage efficiency is request scheduling [7-9]. However, since the mechanism to access the storage medium is different from hard disks, a different approach is required for MEMS-based storage. Specifically, thousands of parallel-activating heads make the scheduling problem in MEMS-based storage even more complicated when considering the potential performance of the parallel processing ability.

In this paper, we discuss three requirements of scheduling algorithms for MEMS-based storage, namely short positioning delay, high-parallelism, and low variation of service latency. In terms of these requirements, we compare various versions of I/O scheduling algorithms for MEMS-based storage and then show the effectiveness of parallelism-aware scheduling algorithms. Simulation results show that the parallelism-aware request scheduling algorithms

perform better than SPTF (Shortest Positioning Time First) in terms of the average request delay when the workload is sufficiently heavy.

The remainder of the paper is organized as follows. Section 2 explains basic structure of MEMS-based storage and gives an overview of scheduling algorithms for MEMS-based storage. We present parallelism-aware request scheduling algorithms in Section 3, and show the experimental results in Section 4. Finally, Section 5 presents the conclusion of this paper.

2 Related Works

2.1 Basic Structure of MEMS-based Storage

A MEMS-based storage device consists of the magnetic media (called *media sled*) that is divided into regions and groups of heads (called *probe tips*) used to access data on the corresponding region. To access data on a specific (x, y) location, MEMS-based storage suffers a substantial distance-dependent positioning time delay similar to disks. Unlike disks, however, the heads of MEMS-based storage are fixed and magnetic media itself moves to access data on a specific location. The movement of the media in the directions of x and y axes is independent and proceeds in parallel. Thus, the positioning time $time_{position}(x, y)$ for a specific (x, y) location can be calculated as follows.

$$time_{position}(x, y) = max\ (\ time_{seek\_x}, time_{seek\_y}\ ) \qquad (1)$$

where $time_{seek\_x}$ and $time_{seek\_y}$ are the seek times on the x and y dimensions, respectively. In most current architectures, $time_{seek\_x}$ is dominant over $time_{seek\_y}$ because extra settling time must be included to $time_{seek\_x}$, but not to $time_{seek\_y}$. Settling time is the time needed for the oscillations of the magnetic media to damp out. This time is dependent on the construction of the magnetic media and the stiffness of the spring that sustains the magnetic media [4]. Since media access is performed in the direction of the y dimension after positioning, it requires constant media velocity in the y dimension and zero velocity in the x dimension. Hence, oscillation in the x dimension leads to off-track interference after seeking, while the same oscillation in the y dimension affects only the bit rate of the data transfer [6].

2.2 Basic Scheduling Algorithms

Scheduling algorithms such as FCFS, which services requests in the order of arrival, SSTF, which services requests from smallest to largest seek time [10], and SPTF, which services requests from the smallest positioning time, that is, the delay considering both seek time and rotational latency, to the largest [5], have been suggested for conventional disks. Griffin et al. showed that all of these scheduling algorithms work appropriately when applied to MEMS-based storage [2] by mapping seek time to the movement delay on the x-axis direction and rotational latency to

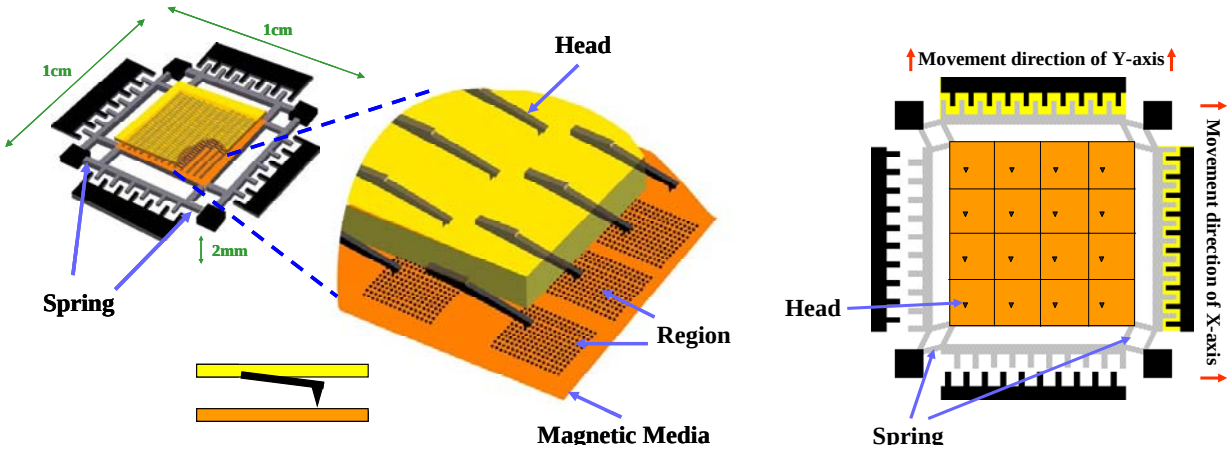


Fig. 1. Physical structure of a MEMS-based storage device [3]. There are thousands of regions on the magnetic media and a read/write head for each corresponding region. The magnetic media moves along two directional axes, x and y.

the movement delay on the *y*-axis direction.

Some recent studies suggested scheduling algorithms for MEMS-based storage considering the physical characteristics of MEMS devices [1, 3]. Yu et al. suggested a minimum spanning tree based scheduling algorithm for MEMS-based storage [3]. Since SPTF does not perform well in terms of average response time for some cases due to its greedy property, their new algorithm generates a minimum spanning tree (MST) for pending requests based on seek distance and schedules the requests by traversing the tree in double walking order. They show that the MST-based algorithm shows slightly better performance than SPTF in terms of average response time. However, this algorithm incurs additional overhead in making and rebuilding the MST whenever a new request arrives. Furthermore, this algorithm inherits the starvation problem similar to SPTF and SSTF, in that some requests may be delayed indefinitely.

Hong et al. suggested the ZSPTF (Zone-based Shortest Positioning Time First) scheduling algorithm [1, 17]. They considered the starvation problem of SPTF. ZSPTF groups the regions of the storage media into a set of zones based on seek time equivalence. Zones are serviced by the C-SCAN (Circular Scan) order, and within a zone, requests are serviced by the SPTF order. ZSPTF performs a little worse than SPTF in terms of average response time, but performs better in terms of the squared coefficient of variation of response times which means that fairness is improved and the chance of starvation lessened.

Lee et al. [15, 16] presented a parallelism-aware request scheduling algorithm called P-SPTF for MEMS-based storage devices. P-SPTF assigns a priority value to each (x, y) position in a region and the position with the highest priority value is scheduled first. The idea of the algorithm is to assign higher priority to the position that contains the most number of requests in each (x, y) position. This allows a large number of waiting requests to be simultaneously serviced first, leading to improved average response time. They also suggested another algorithm, called PA-SPTF, that incorporates the aging factor to the P-SPTF algorithm, making it starvation-free.

3 Scheduling Algorithms for High-Parallelism MEMS-based Storage

Scheduling algorithms of MEMS-based storage for heavy I/O workload systems should consider the following three problems. First, the algorithm should exploit the parallelism of MEMS-based storage. Since thousands of heads can work simultaneously, it is more efficient to provide higher priorities to those locations that have large number of pending I/O requests. Second, the algorithm should consider the positioning time of heads. Specifically, minimization of head movement is important to service pending requests in the scattered (x, y) positions efficiently. Third, the algorithm should consider the fairness issue. Even though efficiency is important, there may be the starvation problem of requests in some specific locations. The first two requirements are related to the efficiency of the scheduling, and the third one is related to fairness. In Subsection 3.1, we present the efficiency of scheduling and then we present the fairness issue of the scheduling in Subsection 3.2.

3.1 Efficiency Issue of Scheduling

The parallelism-aware scheduling algorithm exploits the fact that there may be pending requests with identical relative x and y offsets in multiple regions and that MEMS-based storage can handle these requests simultaneously. Using this characteristic, the scheduling algorithm could improve the average response time by assigning higher execution priority to the position that has the most number of requests in each (x, y) position, which we denote as $N(x, y)$.

This parallelism-aware scheduling algorithm can be combined with any positioning time aware scheduling algorithm such as SSTF or SPTF, which we refer to as the positional scheduling algorithm. For the rest of the paper, we will use the SPTF scheduling algorithm as the positional scheduling algorithm. With these two properties, P-SPTF algorithm is presented [15, 16]. Priority in P-SPTF is determined as follows. Let $Priority(x, y)$ be the priority value to determine the next request to service. Then, $Priority(x, y)$ is computed as

$$Priority(x, y) = N(x, y) / time_{position}(x, y) \qquad (2)$$

where $N(x, y)$ is the number of pending requests on position (x, y) across all regions in the device and $time_{position}(x, y)$ is the positioning time from the current position to position (x, y) as determined by SPTF. The request which has the largest value of $Priority(x, y)$ is selected as the next location to be serviced.

3.2 Fairness Issue of Scheduling

Fairness is an important factor that needs to be considered for request scheduling algorithms. Greedy algorithms that only consider the high-parallelism or short positioning time may not be adequate in terms of fairness as it may lead to starvation of some requests. For the SPTF or P-SPTF algorithm, this may happen when a request is far away from the current position or there are relatively fewer requests that can be processed simultaneously. It is likely that eventually the request will be serviced. However, the response time for that request can have far larger variance than the average cases. This implies that some requests are receiving faster service at the expense of others that may be suffering starvation.

To alleviate this problem, an aging factor may be introduced. Lee et al. suggested the PA-SPTF algorithm that incorporates the aging factor into P-SPTF. The aging factor considered in PA-SPTF is the waiting time of the requests. PA-SPTF determines the request to be serviced using a priority value similar to P-SPTF. The scheduling priority $Priority(x, y)$ of each location is determined as

$$Priority(x, y) = \sum W(x, y) / time_{position}(x, y) \qquad (3)$$

where $time_{position}(x, y)$, likewise as in P-SPTF, is the positioning time from the current position to position (x, y) and $\sum W(x, y)$ represents the sum of the waiting times for requests on the position (x, y). $\sum W(x, y)$ incorporates the notion of the number of pending requests on location (x, y) as well as the aging factor.

There is another way to consider the fairness of the scheduling that can be used for starvation resistant. The basic idea is to traverse all positions of the region during each sweep similar to the SCAN algorithm in hard disks.

The ZSPTF (zone-based SPTF) algorithm presented by Hong et al. [1, 17] can be classified into this category. ZSPTF divides each region into several rectangular zones. Then, it first services requests in a zone by the SPTF algorithm and then moves to another zone chosen by the C-SCAN manner. This algorithm resolves the starvation problem of SPTF that may happen to some requests far from the current head position. However, ZSPTF does not consider the parallelism of the MEMS-based storage.

3.3 Combining Efficiency and Fairness

Through Subsections 3.1 and 3.2, we have discussed three requirements of the scheduling algorithm in MEMS-based storage. The first two requirements, namely short positioning time and high-parallelism, are related to efficiency and the third one, low variation of service latency, is related to fairness.

SSTF and SPTF consider only the first requirement. P-SPTF considers the first and the second requirements while ZSPTF considers the first and the third requirements. PA-SPTF considers all of these three requirements.

In this subsection, we present yet another two scheduling algorithms, called ZSCAN-PSPTF and ZPSPTF, that combines the idea of P-SPTF and ZSPTF. Unlike PA-SPTF that combines all requirements in a single priority measure, ZSCAN-PSPTF and ZPSPTF use the granularity of scheduling similar to ZSPTF.

Now, let us look at the algorithm details of the two algorithms. The first algorithm, ZSCAN-PSPTF, divides each region into zones similar to ZSPTF. It first selects a zone and then services all requests in the zone and moves to another zone. Inter-zone scheduling is performed by the SCAN order and intra-zone scheduling is performed by the P-SPTF order. The second algorithm, ZPSPTF, uses P-SPTF for both inter-zone scheduling and intra-zone scheduling.

4 Experimental Results

To assess the effectiveness of the scheduling algorithms, we have performed trace-driven simulations. A typical logical block of 512 bytes is mapped to 8 byte sectors of MEMS-based storage at the same relative position in 64 different regions, which are accessed concurrently. Adjacent logical blocks are allocated sequentially to the *y*-axis direction to allow for successive access without repositioning.

In the traces used in our experiments, the inter-arrival times of requests conform to an exponential distribution. The request size is also exponential with a mean of 4KB, and the placement of requests is uniformly distributed across the entire device.

To explore a range of workload intensities, we scale the traced inter-arrival times to produce a range of average inter-arrival times. For example, a scaling factor of two generates a workload that is twice more intense than the original trace. When the total size of distinct blocks in the trace is larger than the capacity of

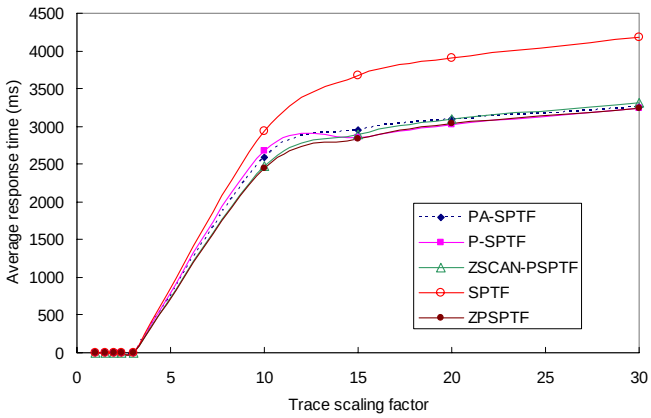


Fig. 2. Average response time of the algorithms as the trace scaling factor increases.

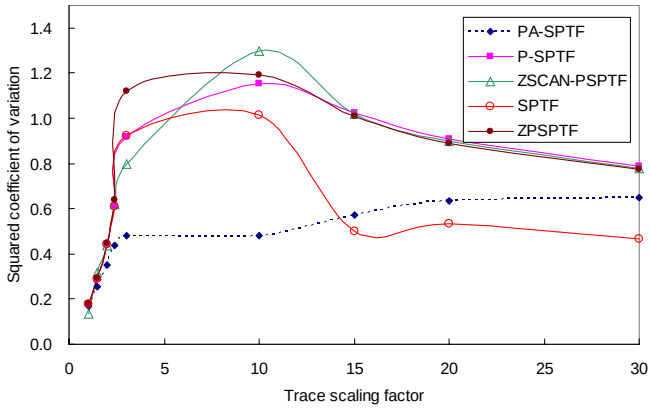


Fig. 3. Squared coefficient of variation of response times as the trace scaling factor increases.

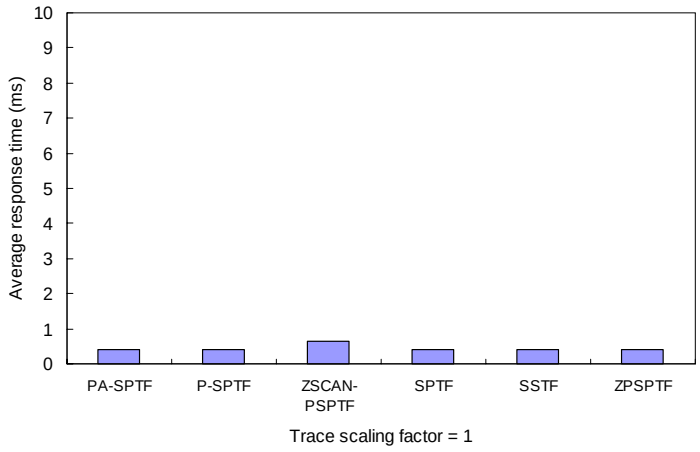
a single MEMS device, we used multiple media sleds. The sleds move simultaneously and their relative positions are unchanged.

We compare P-SPTF, PA-SPTF, SPTF, ZPSPTF and ZSCAN-PSPTF. For ZPSPTF and ZSCAN-PSPTF, we used 25×3 zones for each region.

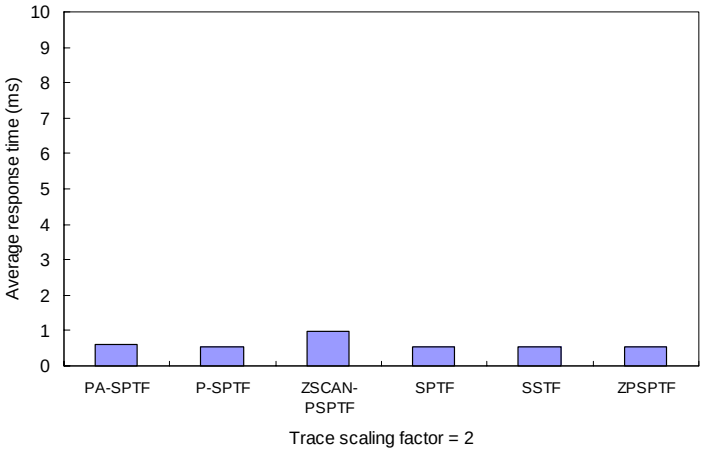
Fig. 2 shows the average response times as a function of the trace scaling factor for the five algorithms. The shape of the graph is similar to the results presented by Lee et al. [15, 16] but precise values are somewhat inconsistent due to different configurations of the storage device.

As can be seen from Fig. 2, parallelism-aware scheduling algorithms P-SPTF, PA-SPTF, ZPSPTF, and ZSCAN-PSPTF perform better than SPTF by a large margin when the workload intensity becomes sufficiently heavy. This is because parallelism-aware scheduling algorithms consider the number of requests at identical relative positions for request scheduling. Rapid service of a large number of pending requests leads to the smaller average response time. Based on these results, we can conclude that parallelism-aware scheduling algorithms are more scalable than other algorithms and can be effective when employed to large server environments with heavy request streams such as multimedia and Web servers. However, there are little performance differences among parallelism-aware scheduling algorithms such as P-SPTF, PA-SPTF, ZPSPTF, and ZSCAN-PSPTF.

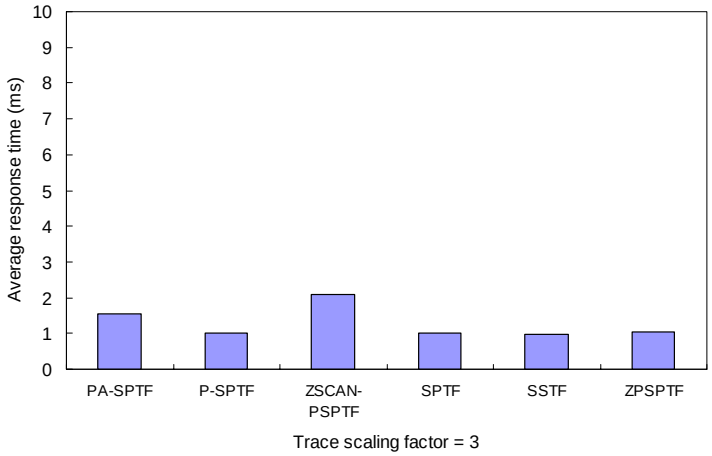
Fig. 3 shows the squared coefficients of variation of response times (σ^2/μ^2) as the trace scaling factor increases, where σ is the standard deviation of response times and μ is the average response time. For this metric, a lower value means that the response time for each request deviates less from the average case. It is a metric used to measure starvation resistance and fairness [5]. By incorporating the aging factor via the sum of the waiting time, PA-SPTF shows comparable performance with other algorithms for all ranges of the experiments.



(a) Average response time when the trace scaling factor is 1.

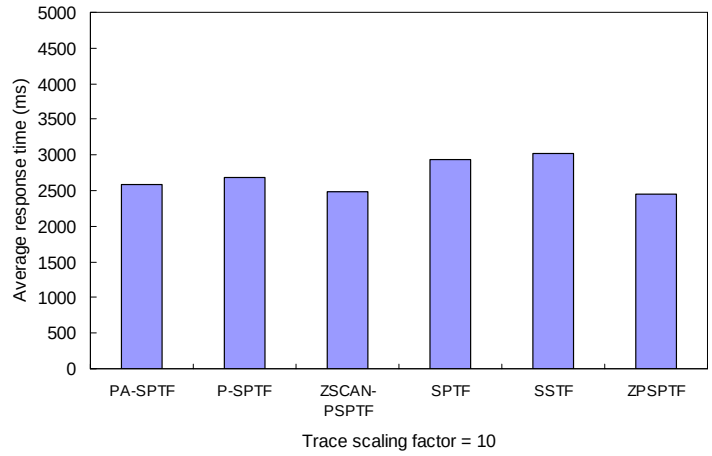


(b) Average response time when the trace scaling factor is 2.

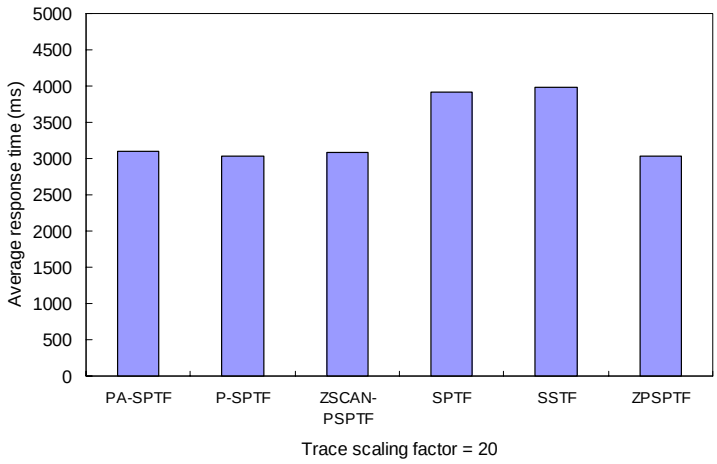


(c) Average response time when the trace scaling factor is 3.

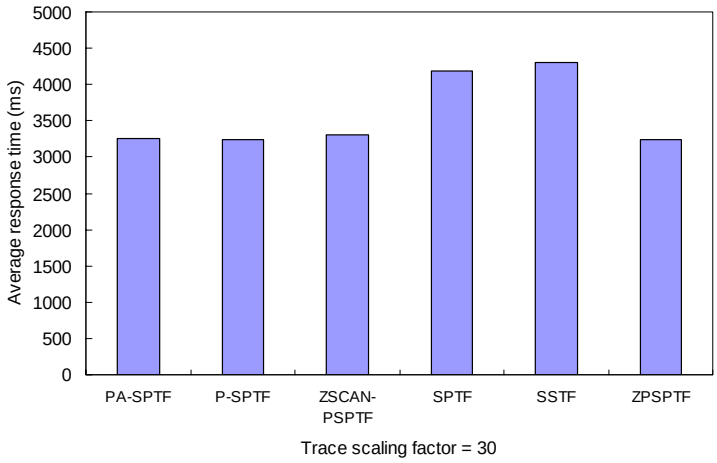
Fig. 4. Average response time of the algorithms when the trace scaling factor is small.



(a) Average response time when the trace scaling factor is 10.

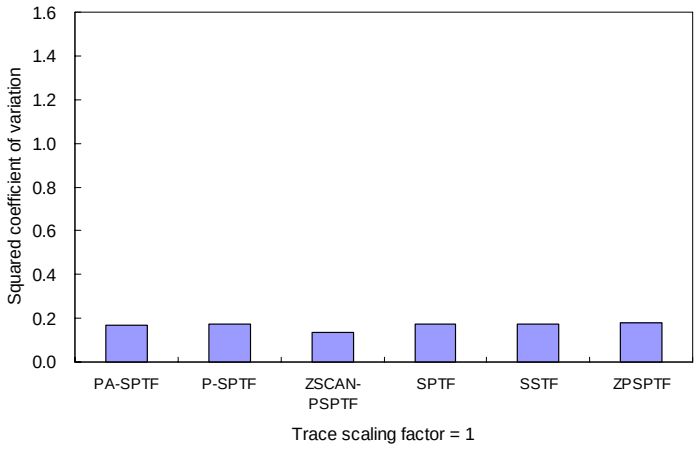


(b) Average response time when the trace scaling factor is 20.

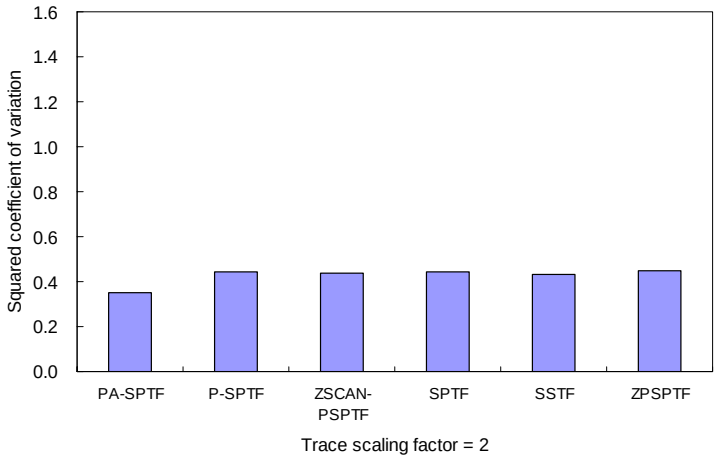


(c) Average response time when the trace scaling factor is 30.

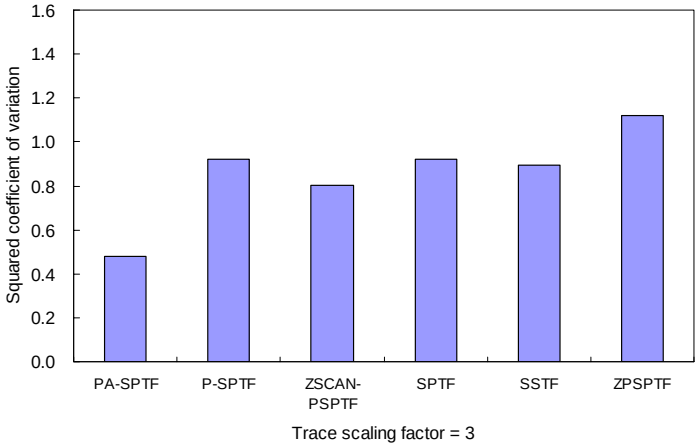
Fig. 5. Average response time of the algorithms when the trace scaling factor is large.



(a) Squared coefficient of variation when the trace scaling factor is 1.

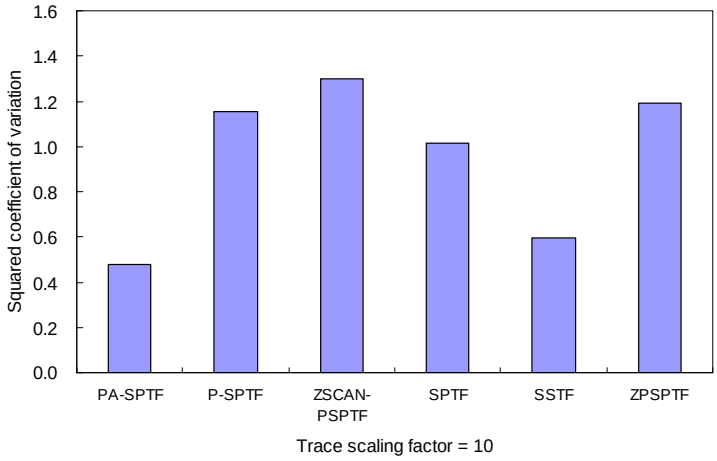


(b) Squared coefficient of variation when the trace scaling factor is 2.

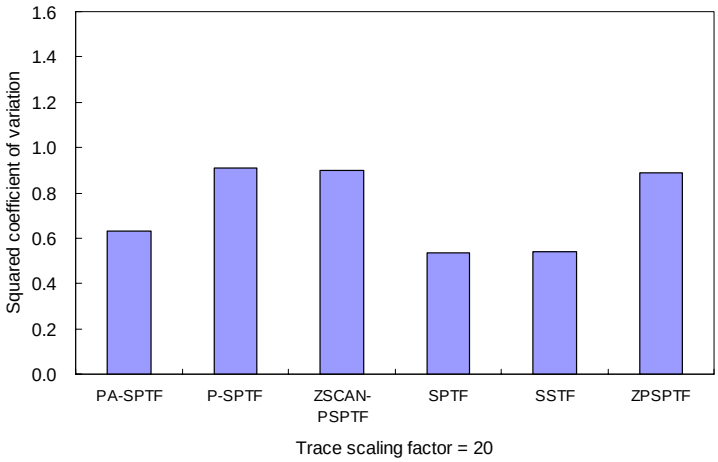


(c) Squared coefficient of variation when the trace scaling factor is 3.

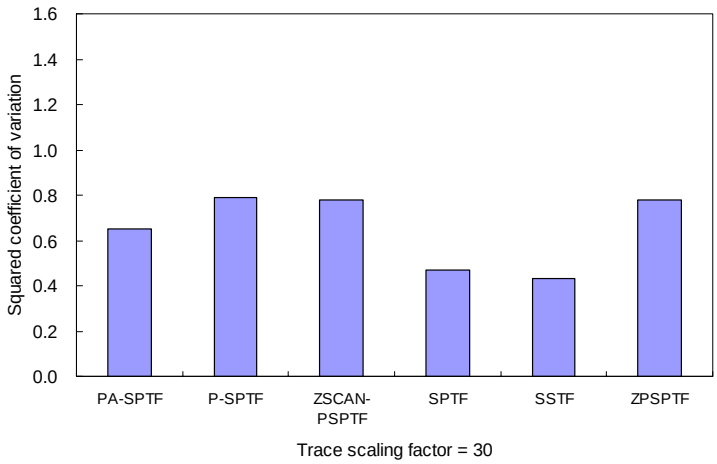
Fig. 6. Squared coefficient of variation of response times when the trace scaling factor is small.



(a) Squared coefficient of variation when the trace scaling factor is 10.



(b) Squared coefficient of variation when the trace scaling factor is 20.



(c) Squared coefficient of variation when the trace scaling factor is 30.

Fig. 7. Squared coefficient of variation of response times when the trace scaling factor is large.

Zone-based parallelism-aware algorithms such as ZSCAN-PSPTF and ZPSPTF do not perform well in terms of the squared coefficients of variation of response times in our experiments. Since ZPSPTF selects a zone to schedule by the P-SPTF order, the variation of response time can be large. However, it is not expected that ZSCAN-PSPTF shows large variation of response time because it uses SCAN as the inter-zone scheduling algorithm. With a finer-grained zone size, we could improve the squared coefficient of variation of response time for ZSCAN-PSPTF. However, a smaller zone size deteriorates the average response time because of lower-parallelism. Figs. 4-7 show the performance results separately for each trace scaling factor.

5 Conclusion

In this paper, we discussed three requirements of scheduling algorithms for MEMS-based storage, namely short positioning delay, high-parallelism, and low variation of service latency. In terms of these requirements, we compared various versions of parallelism-aware I/O scheduling algorithms for MEMS-based storage and then showed the effectiveness of them through trace-driven simulations. Experimental results show that parallelism-aware scheduling algorithms perform better than SPTF in terms of the average request delays when the workload is sufficiently heavy.

6 Acknowledgements

This work has been supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2008-314-D00344).

References:

[1] B. Hong, S. Brandt, D. Long, E. Miller, K. Glocer, and Z. Peterson, "Zone-based Shortest Positioning Time First Scheduling for MEMS-based Storage Devices," *11th IEEE/ACM Symp. Modeling, Analysis, and Simulation of Computer and Tel. Systems*, 2003.

[2] J. Griffin, S. Schlosser, G. Ganger, and D. Nagle, "Operating system management of MEMS-based storage devices," *4th USENIX Symp. Operating Systems Design and Implementation*, pp. 227-242, 2000.

[3] H. Yu, D. Agrawal, and A. Abbadi, "Towards optimal I/O scheduling for MEMS-based storage," *20th IEEE/11th NASA Goddard Conf. Mass Storage Systems and Technologies*, 2003.

[4] J. Griffin, S. Schlosser, G. Ganger, and D. Nagle, "Modeling and performance of MEMS-based storage devices," *ACM SIGMETRICS Conf.*, pp. 56-65, 2000.

[5] B. Worthington, G. Ganger, and Y. Patt, "Scheduling Algorithms for Modern Disk Drives," *ACM SIGMETRICS Conf.*, pp. 241-251, 1994.

[6] S. Schlosser and G. Ganger, "MEMS-based storage devices and standard disk interfaces: A square peg in a round hole?" *3rd USENIX Conf. File and Storage Technologies*, 2004.

[7] M. Liu and K. Gao, "High Efficient Scheduling Mechanism for Distributed Knowledge Discovery Platform," *WSEAS Transactions on Information Science and Applications*, Vol.6, No.1, 2009.

[8] J. Li, "A Sampling-based Method for Dynamic Scheduling in Distributed Data Mining Environment," *WSEAS Transactions on Computers*, Vol.8 No.1, 2009

[9] A. J. S. Santiago, A. J. Yuste, J. E. M. Exposito, S. G. Galan, J. M. M. Marin, and S. Bruque, "A Dynamic-Balanced Scheduler for Genetic Algorithms for Grid Computing," *WSEAS Transactions on Computers*, Vol.8, No.1, 2009.

[10] P. Denning, "Effects of scheduling on file memory operations," *AFIPS Spring Computer Conf.*, pp.9-21, 1967.

[11] S. Schlosser, J. Griffin, D. Nagle, and G. Ganger, "Designing computer systems with MEMS-based storage," *9th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2000.

[12] P. Vettiger, M. Despont, U. Drechsler, U. Dürig, W. Häberle, M. Lutwyche, H. Rothuizen, R. Stutz, R. Widmer, and G. Binnig, "The Millipede – More than one thousand tips for future AFM data storage," *IBM Journal Research and Development*, Vol.44, No.3, pp.323-340, 2000.

[13] R. Rangaswami, Z. Dimitrijevic, E. Chang, and K. Schauser, "MEMS-based disk buffer for streaming media servers," *Int'l Conf. Data Engineering*, 2003.

[14] H. Yu, D. Agrawal, and A. Abbadi, "Tabular placement of relational data on MEMS-based storage devices," *Int'l Conf. Very Large Databases*, 2003.

[15] H. Bahn, S. Lee, and S. H. Noh, "P/PA-SPTF: Parallelism-aware Request Scheduling Algorithms for MEMS-based Storage Devices," *ACM Transactions on Storage*, Vol.5, No.1, 2009.

[16] S. Lee, H. Bahn, and S. H. Noh, "Parallelism-aware Request Scheduling for MEMS-based Storage Devices," *14th IEEE Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2006.

[17] B. Hong, S. Brandt, D. Long, E. Miller, K. Glocer, and Z. Peterson, "Using MEMS-based Storage in Computer Systems – Device Modeling and Management," *ACM Transaction on Storage*, Vol. 2, No.2, pp 139-160, 2006.