ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 5, No. 5, 660-668 (2020)

www.astesj.com

Special Issue on Multidisciplinary Sciences and Engineering

ASTESJ

ISSN: 2415-6698

Issues in File Caching and Virtual Memory Paging with Fast SCM Storage

Yunjoo Park, Hyokyung Bahn*

Department of Computer Science & Engineering, Ewha Womans University, 03760, South Korea

ARTICLE INFO

Article history:
Received: 09 August, 2020
Accepted: 21 September, 2020
Online: 05 October, 2020

Keywords:
Storage-Class Memory (SCM)
File Caching
Virtual Memory Paging

ABSTRACT

Storage-Class Memory (SCM) like Optane™ has advanced as a fast storage medium, and conventional memory management systems designed for the hard disk storage need to be reconsidered. In this article, we revisit the memory management system that adopts SCM as the underlying storage medium and discuss the issues in two layers: file caching and virtual memory paging. Our first observation shows that file caching in the SCM storage is profitable only if the cached data is referenced more than once, which is different from the file caching in hard disks, where a single hit is also beneficial. Our second observation in virtual memory paging shows that the page size in the SCM storage is sensitive to the memory system performance due to the influence of memory address translation and storage access cost. Our simulation studies show that the performance of paging systems can be improved by adjusting the page size appropriately considering application characteristics, storage types, and available memory capacities. However, the page size will not be a significant issue in mobile platforms like Android, where applications are killed before the memory space is exhausted, making situations simpler. We expect that the analysis shown in this article will be useful in configuring file caches and paging systems with the emerging SCM storage.

1. Introduction

With the large performance gap between hard disk drive (HDD) and dynamic random-access memory (DRAM), the main purpose of memory management in computing systems has been the minimization of disk I/Os [1, 2]. The access latency of hard disks is more than tens of milliseconds, which is five to six orders of magnitude larger than DRAM's access latency. Meanwhile, due to the rapid improvement of storage access time by the adoption of flash-based solid state drive (SSD) and storage-class memory (SCM), the extremely large performance gap has been decreased [3-5]. The access latency of the flash storage is less than fifty milliseconds, and hence the performance gap of storage and memory becomes less than 3 orders of magnitude. Such trends have been speeded up by the commercialization of SCM whose access latency is just 1 or 2 orders of magnitude slower than DRAM [6, 7].

A lot of patents related to the detailed architectures and algorithms of SCM management have been suggested, and Intel manufactured the commercial product of SCM, called Optane™ [8, 9]. Owing to its desirable features like high performance, low energy consumption, and long write endurance, SCM is anticipated to be adopted in the storage systems like flash SSD and hard disks [10-13].

SCM can also be adopted in the main memory system because it allows byte-accesses like DRAM but consumes less energy because it is a non-volatile medium [14]. However, the access latency of SCM is longer than that of DRAM, and hence it is now considered as high-end storage or additional memory that can be used together with DRAM. Although SCM may be used as either memory or storage, this article focuses on storage. Since the performance gap of storage and memory becomes small by adopting SCM, memory management systems targeting at slow hard disk storage need to be revisited.

In this article, we quantify the performance of systems based on SCM storage and analyze a couple of issues in the management of main memory under the SCM-based storage. In particular, this article analyzes two memory management hierarchies affected by the acceleration of storage devices, *file caching* and *virtual memory paging*.

File caching preserves file data read from the storage to the main memory area called the file cache and services requests for the same data from the cache without accessing storage. The

*Corresponding Author: H. Bahn, 52, Ewhayeodae-gil, Seodaemun-gu, Seoul 03760, Republic of Korea, +82-2-3277-4247, bahn@ewha.ac.kr.

purpose of file caching is to minimize the storage access frequency. However, since the storage access latency is fast enough by making use of SCM, it is questionable whether file caching is still needed. To validate this, we investigate the condition that file caching is effective according as the storage access latency is varied. Our preliminary study exhibits that a storage access takes about 30% more latency than a cache access even though the same data is accessed and the access latency of storage and cache is identical. This is because there exists heavy software stack to passing through the storage media. By observing this result, our finding is that file caching is still needed for SCM-based storage if the data in the cache is requested more than once after inserted into the cache. That is, since file caching needs additional time for inserting the requested data into the cache, more than a certain cache hits are necessary for caching to be profitable, and we show that this is at least twice for SCM storage. Note that low-end storage media like hard disks require only a single hit for caching to gain.

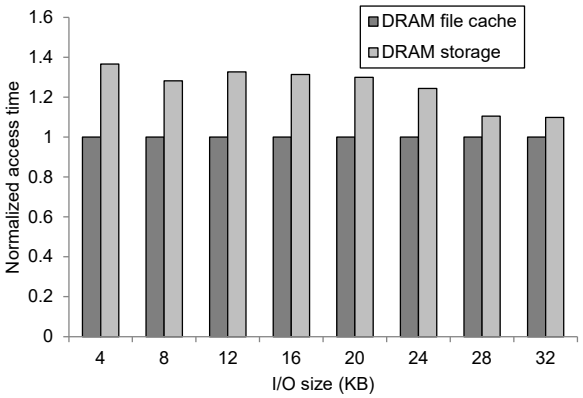


Figure 1: Latency of accessing data from the file cache and the DRAM-based storage.

Our second observation focuses on virtual memory paging if SCM is made use of as storage media. Similar to file caching, the purpose of virtual memory paging is the minimization of storage accesses, which is also called page faults, when storage is hard disk. Since the storage medium becomes fast enough, we observe that page fault handling may not be the main performance bottleneck of virtual memory paging. In particular, as the latency of storage access becomes small by adopting SCM, the bottleneck of the page access latency may be shifted to memory address translation. Note that to access a memory page, translating the address should be done first and then the data in the page is accessed from either memory or storage. As the data access is accelerated by the reduced storage access cost, the address translation process may be a new bottleneck by accessing page tables.

This article quantifies what will be the main bottleneck of virtual memory paging as the storage performance and the page size are changed. Based on our analysis, we discover the following two phenomena. First, a small page is not efficient in terms of the page fault rate but it performs well in terms of the data access latency. The reason is that a page fault handling latency is strongly related to the page size when the storage is SCM, which does not need seek movement. Second, even though a small page is efficient with respect to the latency of data access, it degrades the latency of address translation by increasing TLB miss counts. Due to this reason, deciding a page size needs to consider the trade-off relation of the data access latency and the address translation latency.

Our simulation results exhibit that the performance of virtual memory paging can be improved by adjusting the page size appropriately considering application characteristics, storage types, and available memory sizes. However, the page size will not be a significant issue in mobile platforms like Android, where applications are killed before memory space is exhausted, making situations simpler. We expect that the analysis shown in this article will be useful in configuring file caches and paging systems with the emerging SCM storage.

The remaining part of this article is organized as follows. In Section 2, we quantify the performance implication of file caching as the storage medium changes from hard disk to SCM. Section 3 anatomizes the virtual memory paging performances with high-performance SCM storage particularly focusing on page sizes. Section 4 presents the experimental results through conducting simulation experiments to observe the implications of SCM storage based memory management systems. Section 5 discusses the adoption of our model to the mobile application environments. Finally, we present the conclusion of this article in Section 6.

2. File Caching for SCM Storage

This section quantifies the efficiency of file caching as the storage medium changes from hard disk to SCM. To this end, we measure the latency of file system operations when DRAM is used as storage. Note that DRAM is volatile, but we use a certain area of DRAM as a storage partition just to see the effect of fast storage media. Note also that this situation implies the optimistic performance of SCM storage. We added a profiler to Ext4 for measuring the latency of directly accessing data from the DRAM storage and the latency of accessing data in the DRAM file cache.

Figure 1 depicts the access latency of DRAM storage and DRAM file cache as the size of data accesses changes. In this graph, we measure each case 10 times and plot their average. As can be seen in the graph, the access latency from DRAM storage is 30% longer than accessing the same data from DRAM file cache. Although the same DRAM is used for cache and storage, the performance gap occurs due to the existence of software I/O stack. When considering these results, file caching can be still necessary to buffer the latency gap between storage and memory although their access latencies are identical. However, as the gap is very small, some conditions need to be met for file caching to be beneficial.

When we use file caching, the accessed data should be stored into the cache, which requires additional latency. The access latency of DRAM storage shown in Figure 1 does not include this latency, and hence the accessed data is delivered directly to the user memory. If we use file caching, the accessed data is firstly stored in the file cache and then transferred to the user memory. The overhead of this additional copy operation in memory is not negligible when SCM storage is used because the time overhead to perform a memory copy is similar to that of a storage access. Thus, the gain of file caching is small or there are no profits at all because of this trade-off. Due to this reason, file caching is beneficial only when the benefit of subsequent cache hits is larger than the cost of

the additional memory copy operation. That is, cache hits are important for a cached data to gain, which differs from low-end storage media like hard disk drives where a single hit of data is sufficient for caching to gain. To quantify this, we measure the data access time t from the file cache, storage access time T, and time to access storage including the file caching cost T_m. Then, we evaluate the condition of caching to be beneficial with respect to the cache hit counts. The following equations represent the latency to access data with file cache T_c and without file cache T_x, respectively.

$$T_c = T_m + (n-1)\,t \quad (1)$$

$$T_x = n\,T \quad (2)$$

where n is the total access count for the data. File caching gains when T_c is less than T_x as follows.

$$n > (T_m - t) / (T - t) \quad (3)$$

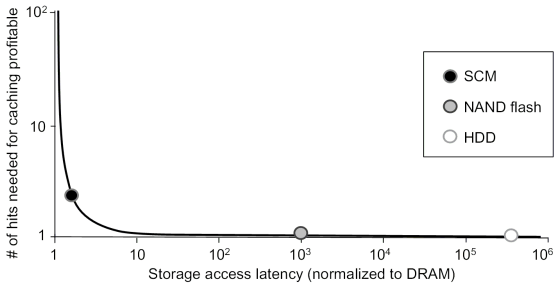
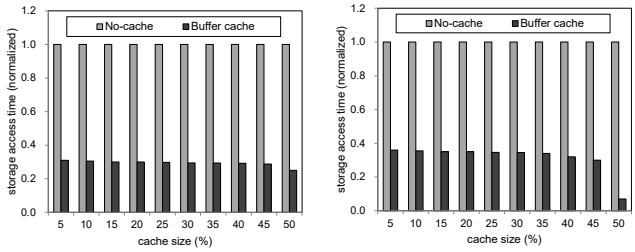


Figure 2: Hit counts necessary for a cached item to be beneficial as a function of the latency for accessing the storage media.

Based on Expression (3), we estimate the cache hit counts needed for caching to be beneficial. Figure 2 depicts the required hit counts when the storage media is flash memory, hard disk, and SCM. As plotted in the figure, the cache hit count necessary for caching to be profitable increases significantly as the performance gap of storage and memory is small. In the case of SCM storage whose access latency is only 30% slower than memory, the cache hit count necessary for caching to be profitable is two while resident in the cache. Hence, if a storage data is requested now and it will not be reused at least twice in the future, it would be better not to store it in the cache for performance improvement.

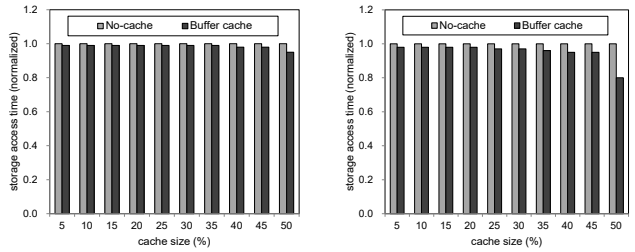
To quantify the influence of file caching in practical situations, we collect file request traces during the execution of two popular storage benchmarks, and replay them as the storage media is varied. The captured traces are web server and proxy server. We investigate the storage access latency with/without file caching under the two workload conditions. For the cache eviction algorithm, we use the least-recently-used (LRU), the most commonly adopted algorithm in file caching. LRU selects the data that was accessed the oldest among all data in the file cache and evicts it if there are no cache spaces to insert new data items.

Figure 3 depicts the storage access time with/without file caching when hard disk storage is used as the size of cache changes. Note that the cache size of 100% means that the cache size is equal to the total footprint of workloads, implying that the eviction algorithm is not necessary. This is not a realistic situation and in practical environments, the cache size is less than 50%. As can be seen from this figure, the effectiveness of file caching is significant in case of the hard disk storage. Specifically, file caching accelerates the storage performance by 70%, which is possible because hard disk is very slower than the file cache, and hence decreasing the disk I/O access counts by file caching is effective.



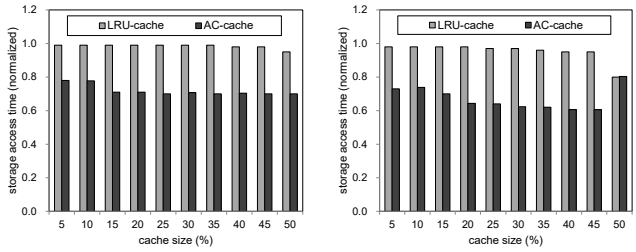
(a) Proxy (b) Web

Figure 3: Effectiveness of file caching under hard disk storage.



(a) Proxy (b) Web

Figure 4: Effectiveness of file caching under SCM storage.



(a) Proxy (b) Web

Figure 5: Performance of file caching with/without admission-controll (AC) under SCM.

Figure 4 depicts the storage access time without/with file caching when the storage medium is SCM. As can be seen from this figure, the effectiveness of file caching is insignificant in case of the SCM storage. In particular, the performance improvement by using file caching is less than 3% with the SCM storage. This apparently exhibits that the effectiveness of file caching decreases significantly as the performance gap of storage and memory becomes small. However, we conducted some new experiments, and show that file caching can be still effective by judicious management.

Figure 5 depicts the storage access time with file caching when the SCM storage is used, but we differentiate the result with two management policies. In particular, LRU-cache adopts the same configuration of Figure 4, which stores all data requested in the file cache and evicts the least-recently-used data if free cache space is necessary. The other graph, denoted by AC-cache (admission-controlled cache), is plotted by allowing the insertion of data into the file cache only if it is used more than once. This allows only

data used multiple times to be cached, thereby filtering out useless single-accessing data. Even though this is a simple technique, the effectiveness of AC-cache is significant as shown in Figure 5. The performance improvement ranges 20-40%. The implication of this result is that file caching is still useful when SCM storage is used, but management policies should be appropriately devised in order to get effective results.

3. Virtual Memory Paging for SCM Storage

This section presents the simulation results of virtual memory paging when we use SCM as storage.

3.1. Access Latency of a Virtual Memory Page

To use a memory page in virtual memory systems, translation of memory addresses between logical address and physical address needs to be conducted. This is done by referencing the page table, which is located at main memory. To improve the address translation performances, a certain part of the page table is cached in the Translation Look-aside Buffer (TLB), which is faster than main memory [15]. When a memory page is requested, address translation by TLB is tried first. The page table is referenced if address translation by TLB misses. Then, the memory data is accessed with the translated physical address. However, if the data does not reside in memory, storage should be accessed, which is called a page fault. Assume that the cache miss rate of TLB is R_T, and the page fault rate is R_F. Then the memory access time T_{TOTAL} can be denoted by

$$T_{TOTAL} = T_{ADDR} + T_{DATA} \quad (4)$$

$$T_{ADDR} = (1 - R_T) * t_\varepsilon + R_T * (t_\varepsilon + t_\tau) \quad (5)$$

$$T_{DATA} = (1 - R_F) * t_\tau + R_F * (t_\tau + T_{PF}) \quad (6)$$

where T_{ADDR} is the time required for performing address translation, T_{DATA} is the latency required for data access, t_ε is the latency required for TLB access, t_τ is the latency for accessing main memory, and T_{PF} is the latency for page fault handling including storage access time.

3.2. Eviction Policies

Since the size of TLB entries is fixed, adding an address translation data to TLB requires the eviction of a certain entry when no TLB entry is available. To choose an eviction victim, we adopt the least-recently-used (LRU) eviction policy, which is a representative algorithm adopted in TLB entry eviction [15].

Because the size of free pages in main memory is fixed, another eviction policy is necessary. Specifically, if the page requested is not in memory and should be loaded from storage, but there is no free page in memory, eviction of a page from memory is necessary. We use the second-chance algorithm, which is a popular eviction policy used in virtual memory systems, in our experiments [14].

The second-chance algorithm investigates if a page has been used recently or not by utilizing the reference bit of each page. When a page is used, the reference bit of that page becomes one. If a free page frame is needed, the second-chance algorithm investigates the reference bits of all pages in memory sequentially, and discards the page firstly found with its reference bit of zero. For each page whose reference bit is one while investigation processes, the second-chance algorithm resets the bit to zero, instead of discarding the page from memory. Thus, if a page is not accessed until the next investigation of the second-chance algorithm, it is evicted.

With this basic configurations, we conduct our simulations to quantify the efficiency of memory management systems when we use SCM-based storage.

3.3. Page Size and Prefetching

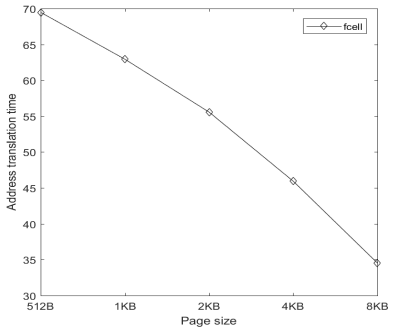
The current operating systems usually manage virtual memory by the unit of *page*. The size of a page is commonly set to 4 kilobytes, which is also the default page size of Linux. However, operating systems also allow the prefetching option, which loads maximum 128 pages together from the secondary storage when a page requested is not in memory. The rationale of this is to consider the characteristics of hard disk, which requires the basic cost for each access consisting of seek latency and rotational latency, accounting for the main part of storage access latency irrespective of the size of data loaded. This implies that loading large data in each request is efficient in case of hard disk based storage. Recent operating systems also support a huge page whose size is up to 4MB. This is also for hard disk storage systems, but it will not be efficient for SCM, in which the page size should be small because storage is fast and there are no seek time or rotational latency.

Nevertheless, it is not feasible to decrease the page size because memory address translation will be efficient with a large page size. That is, TLB with a large page size covers more memory address spaces, leading to improved address translation latency. Then, we need to decide the page size for SCM-based storage by considering the overall effect of storage access and address translation.

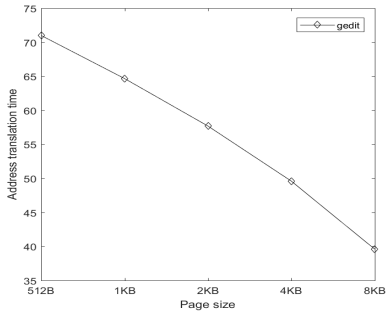
Also, the latency of SCM should be considered, which is optimistically as fast as DRAM, but can be up to 100 or 1000 times slower than DRAM. Due to the variance of performance gap between storage and memory, the relative impact of memory address translation compared to storage access is also varied. In particular, as the performance of SCM approaches that of DRAM, memory address translation incurs relatively more cost. On the contrary, as the performance gap of DRAM and SCM increases, the relative overhead of a storage access will be large. The relation of storage access and address translation will be analyzed in the next section as the performance of SCM is varied.

4. Performance Analysis

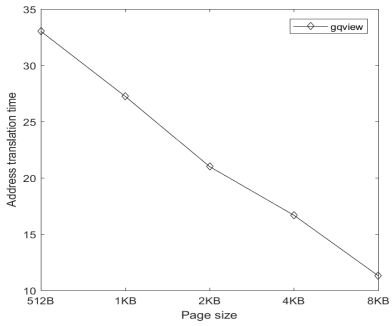
We conduct various simulations to investigate the efficiency of memory management subsystem as SCM storage is adopted. In particular, the impact of the page size on memory performance is investigated. The minimum page size is set to 512 bytes because the size of a page should be at least the block size of the last-level cache memory. We use the virtual memory reference traces captured by the Cachegrind tool of the Valgrind [16, 17]. We collect the virtual memory reference traces from five desktop workloads as listed in Table 1 [18-22].



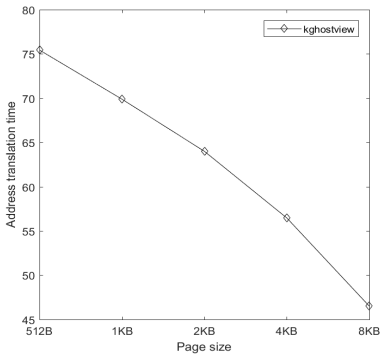
(a) freecell



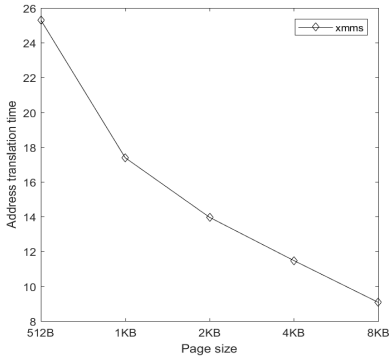
(b) gedit



(c) gqview

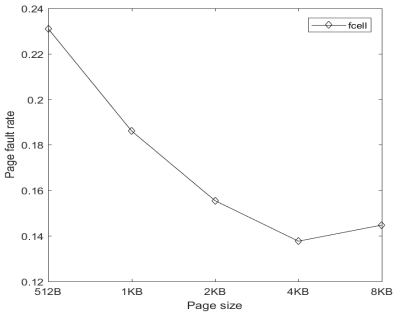


(d) kghostview

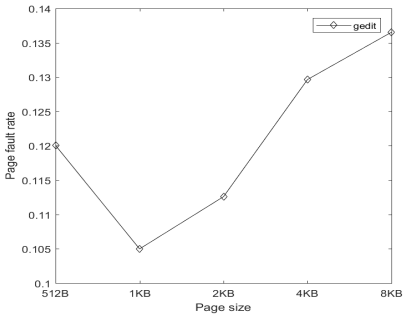


(e) xmms

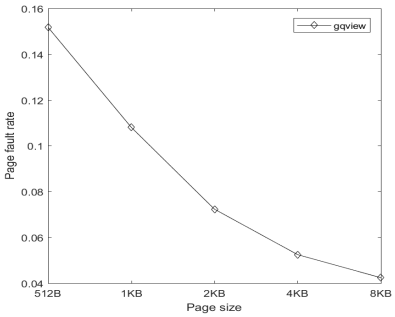
Figure 6: Memory address translation time of desktop workloads as the page size changes.



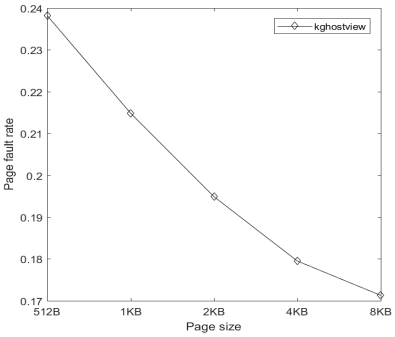
(a) freecell



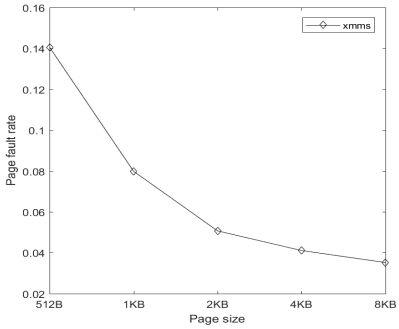
(b) gedit



(c) gqview

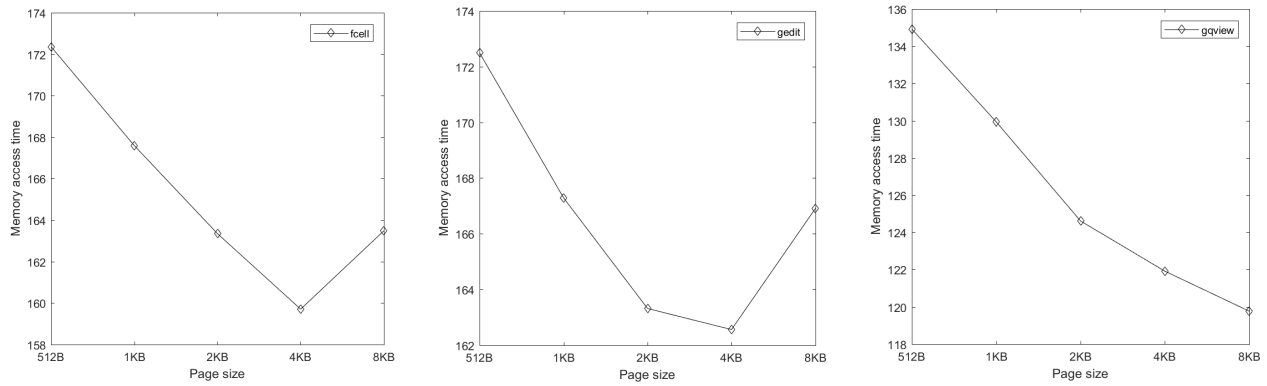


(d) kghostview

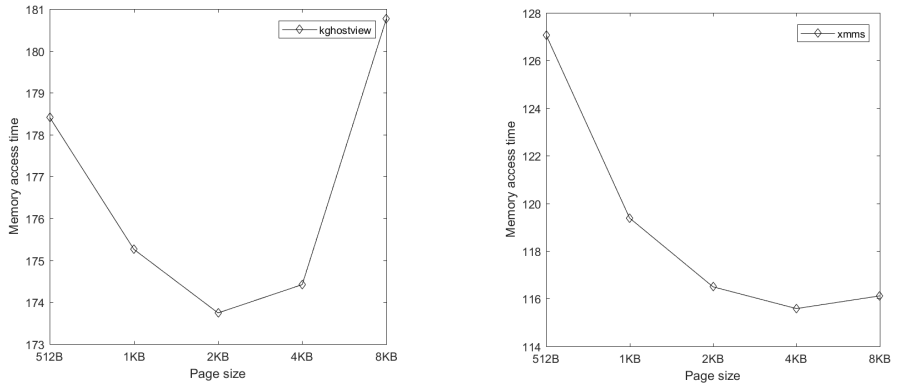


(e) xmms

Figure 7: Page fault rate of desktop workloads as the page size changes.

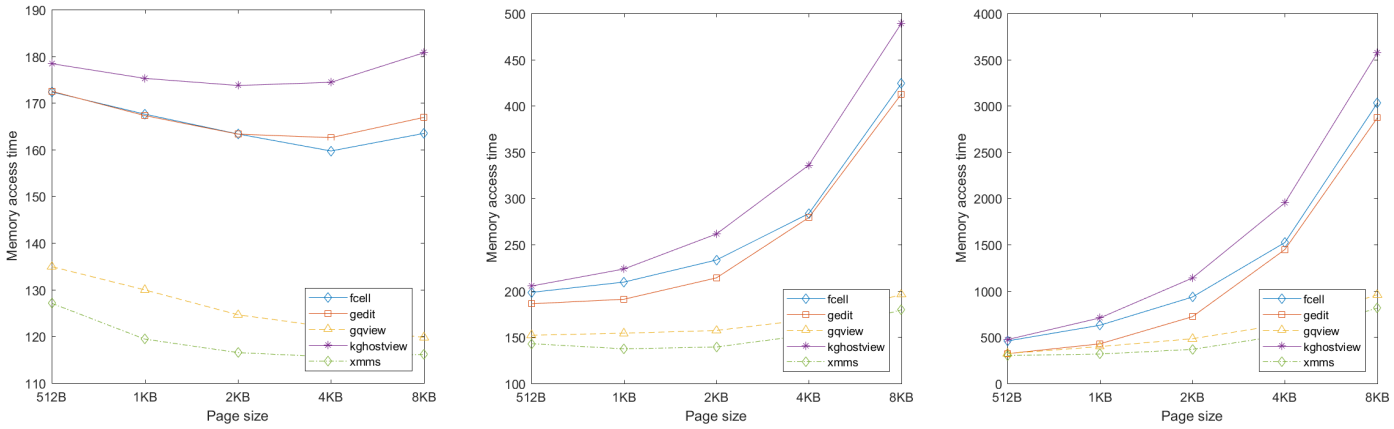


(a) freecell (b) gedit (c) gqview



(d) kghostview (e) xmms

Figure 8: Memory access time of desktop workloads as the page size changes.



(a) 1x of DRAM (b) 10x of DRAM (c) 100x of DRAM

Figure 9: Total memory latency of desktop workloads as the page size and storage access latency change.

4.1. Memory Address Translation

This section discusses the influence of the page size on the performance of memory address translation. Figure 6 depicts the memory address translation latency while running five desktop applications as the page size is changed. As the figure depicts, the latency of memory address translation decreases according as the page size grows. The reason is that the fixed TLB entries account for the address translations of wider memory spaces if the page size becomes large. An increased hit rate of TLB leads to the decreased latency of memory address translation by reducing the number of page table references. Nevertheless, it is known that the hit rate of TLB cannot increase any more when the page size is larger than a certain threshold [23]. Thus, the conclusion of this experiment is that a large page will perform well in terms of the memory address translation, but the page size does not need to be increased any longer after a certain large size.

Table 1: Characteristics of desktop workload

Application	Memory footprint (MB)	Memory access counts		
		Write	Read	Total
freecell [18]	9.84	60,040	430,135	490,175
gedit [19]	14.12	132,822	1,600,941	1,733,763
Gqview [20]	7.26	645,399	265,286	610,685
kghostview [21]	16.98	103,540	1,442,595	1,546,135
xmms [22]	7.86	978,242	190,697	1,168,939

4.2. Storage Access Frequency

This section discusses the impact of the page size on the storage access frequency. Figure 7 depicts the page fault rate of the five applications as the page size changes. As the figure shows, the best page size is not identical for different applications we considered. In most cases, a large page improves the page fault rate. In particular, the page size of 8 kilobytes exhibits the best results for gqview, kghostview, and xmms. On the contrary, the page size that exhibits the best result is 1 kilobytes for gedit and 4 kilobytes for freecell, respectively. The best page size relies not only on the application's characteristics but also on the free memory situation of the total system. When the memory size is insufficient for the given workload environment, decreasing the page size will perform well to retrieve only the data requested at that time. In contrast, when the available memory is sufficient, increasing the page size will be a better choice in terms of the storage access frequency. Also, when the workload is composed of sequential reference patterns, increasing the page size can reduce the number of storage accesses by retrieving large successive data together.

4.3. Total Memory access time

As hard disk requires large seek overhead for each storage access irrespective of the data size, decreasing the storage access frequency can lead to the improvement of data access latency. However, in case of SCM storage, as the data size becomes large, a storage access requires more time to load data. Assume a storage reference stream that is composed of sequential patterns and let us think of the storage access frequency for virtual memory paging that makes use of a large page size and a small page size. It is clear that a large page size is efficient in terms of the storage access frequency, but it does not essentially lead to the enhancement of data access latency because the storage access time becomes large for each I/O with SCM. Hence, instead of improving the storage access frequency, our aim is to reduce the data access latency. In other words, the page fault rate is not a fair performance index for SCM-based virtual memory paging, but the data access latency can be an alternative metric for SCM-based storage.

Unlike the page fault rate case, address translation latency has almost linear relation with the TLB miss rate because the access latency of a page table entry incurred by each TLB miss is identical though the page size is different.

Figure 8 depicts the total memory access latency of the desktop applications we considered as the page size changes. As the figures show, the best page size is not the same for different applications, and is also different from the results of Figure 7 that plots the page fault rate. From this result, we can see that the storage access frequency is not the primary factor of performance in case of SCM storage media. Also, this figure shows that the address translation latency can be another significant factor that influences the memory access time in case of fast SCM storage even though the effectiveness of address translation is smaller than the actual data access. Because a trade-off relation exists between data access and address translation in deciding an appropriate page size, we cannot simply conclude the page size as small or large but a judicious management is necessary for deciding the page size with given environments.

4.4. Relative Storage Performance

Figure 9 depicts the total memory latency for each application as the access latency of storage media changes. Specifically, Figures 9(a), 9(b), and 9(c) represent the total memory latency when the relative access latency of SCM is identical to DRAM, 10 times that of DRAM, and 100 times that of DRAM, respectively. As this figure shows, the page size that performs the best is not the same for each case as the access latency of storage media changes. If the performance of SCM is similar to DRAM as shown in Figure 9(a), the address translation latency becomes important, and hence increasing the page size performs relatively well. In other words, the role of TLB becomes significant in such environments, and the memory access time is improved with the page size of 4KB or 8KB. We cannot determine the best page size for all workloads as it depends on the total memory capacity as well as the workload characteristics. Figures 9(b) and 9(c) plot the total memory latency when the access latency of SCM media is 10 times and 100 times slower than DRAM, respectively. As these two figures show, the memory access time becomes better when we decrease the page size as small as possible. The reason is that the data access latency affects significantly if the storage becomes slow and thus the address translation procedure is less important. The best memory access time can be obtained if we set the page size to 512B for all cases.

Table 2: Characteristics of mobile workload.

Application	Memory footprint (MB)	Memory access counts		
		Write	Read	Total
facebook [24]	198.66	2,045,716	11,607,339	13,653,055
angrybirds [25]	76.94	3,822,479	14,368,068	18,201,717
youtube [26]	68.64	3,162,229	15,034,275	18,196,504
farmstory [27]	53.74	2,101,818	13,122,852	15,224,670
chrome [28]	259.86	4,104,436	16,895,563	20,999,999

When considering these overall situations, we can obtain the best memory access time not by fixing the page size to a constant but by varying relying on the application and storage characteristics. Hence, the common page size of 4KB will not be a good choice for all cases and needs to be changed appropriately if we use the SCM-based fast storage media.

5. Adopting the Model to Android Mobile Applications

In this section, we adopt our model to Android mobile applications to see the effect of the page size in SCM-based storage. To investigate a wide spectrum of Android applications, we collect virtual memory reference traces from five Android applications, namely, facebook a social network service [24], angrybirds a game [25], youtube an online streaming service [26], farmstory an online game [27], and chrome a web browser [28]. Table 2 lists the characteristics of the workloads.

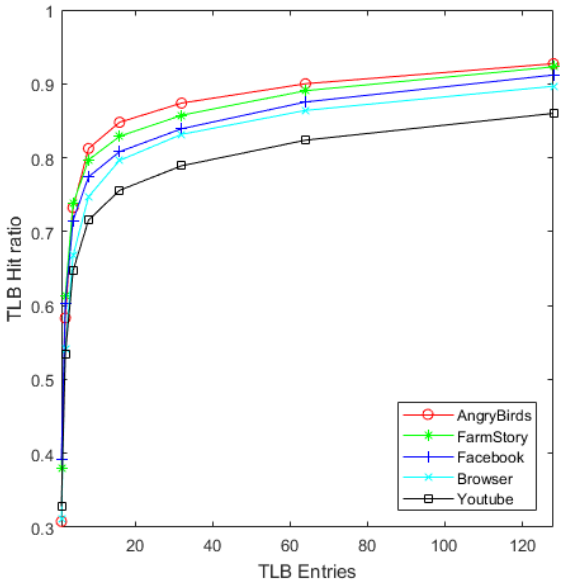


Figure 10: TLB hit rate of mobile workloads as the TLB size changes.

With these memory reference traces, we conduct extensive simulations, and analyze them similar to the desktop system cases in the previous section. Figure 10 depicts the TLB hit rate while executing five mobile applications as a function of the TLB size. As can be seen from the figure, the TLB hit rate is improved significantly as the TLB entries increase. The five applications show similar curve trends. Specifically, the TLB performance is improved sharply when the number of TLB entries is less than 20 and the slope of the curve becomes gradual after that point. Figure 11 depicts the TLB hit ratio as the page size changes. As can be seen from this figure, the TLB hit rate improves according as the page size increases. The reason is that the fixed number of TLB entries account for address translations of more memory capacity as the page size becomes large. An increased hit rate of TLB eventually leads to the decreased latency of memory address translation by reducing the number of page table references. However, the improvement of the TLB hit rate slows down when the page size is larger than a certain threshold. Thus, increasing the page size will perform well in terms of the performance of memory address translation, but it is not sensitive after a certain large page size.

Figure 12 depicts the page fault rate of the five Android applications as the page size changes. As can be seen from the figure, the page fault rate decreases as the page size increases for all applications we considered. Unlike desktop cases that do not exhibit the same trends for different applications, all mobile applications show similar results. The best page size relies on the application's characteristics in desktop cases, but a large page commonly improves the page fault rate of all mobile applications we considered. If the memory space is not sufficient for the given workload environment, large pages will not perform well due to retrieving the data not requested now. However, our Android application experiments show that the memory space is not exhausted in any applications. This is because Android does not use swap but kills applications if free memory space becomes small.

Figure 13 depicts the average memory access time (AMAT) of the five Android applications we considered as the page size changes. This graph separately represents the AMAT when the relative latency of SCM is identical to DRAM, 10 times that of DRAM, 100 times that of DRAM, and 1000 times that of DRAM, respectively. In the previous section, the page size that performs the best in desktop environments is not identical for each case as the performance of storage media is varied. If the performance of storage is similar to that of DRAM, increasing the page size performs relatively well, but if SCM becomes slow, decreasing the page size works well in desktop environments. However, as shown in Figure 13, the best results in mobile environments are obtained when the page size becomes large, and there are no significant differences in memory performances when the page size is larger than 1 kilobytes. This is also related to the memory management policies of Android, in which applications are killed before memory space is exhausted, and thus increasing the page size does not incur problems like desktop environments.

In summary, our conclusion is that the page size will not be a significant issue in mobile environments although fast SCM is used as the storage media.

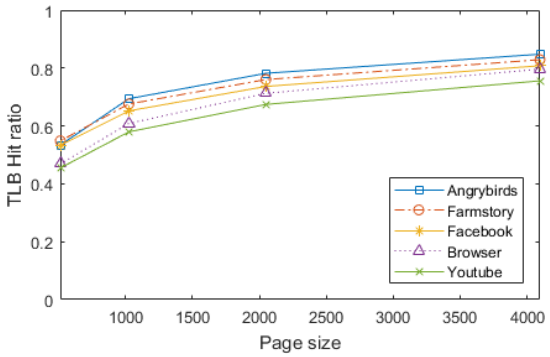


Figure 11: TLB hit rate of mobile workloads as the page size changes.

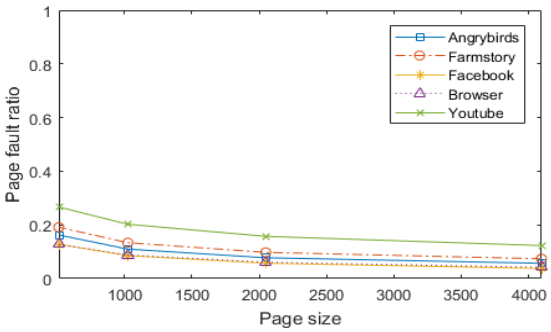


Figure 12: Page fault rate of mobile workloads as the page size changes.

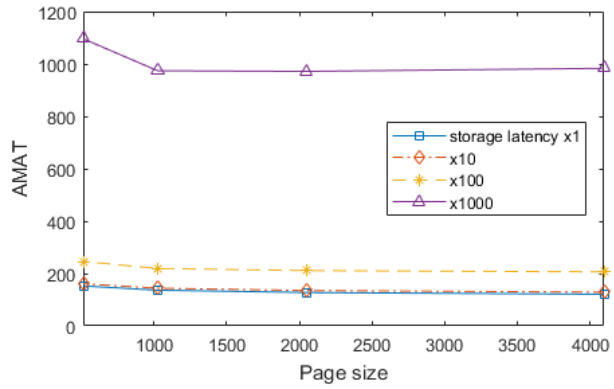


Figure 13: Average memory access time (AMAT) of mobile workloads as the page size and storage latency change.

6. Conclusion

In this article, we considered the effectiveness of traditional memory management systems and configurations if fast storage such as SCM is used as the storage media. Our study discussed the implication of SCM storage with respect to the two important memory/storage subsystems, file caching and virtual memory paging. The analysis results with respect to the file caching showed that caching file blocks is effective if the blocks retrieved from SCM storage is used more than once after loading to memory. Thus, the admission of file blocks to the cache should be controlled for managing the file cache appropriately in SCM-based storage media. In the case of virtual memory paging, our study showed that the number of storage accesses may not be the most important performance index, but the memory address translation process will also become important. In particular, our analysis showed that a common page size of 4 kilobytes is not a good choice for all cases when SCM storage is adopted, and determining the page size by decreasing or increasing it is needed because of the trade-off relation in storage access and memory address translation. However, we also showed that the page size is not a significant issue in mobile platforms like Android, where applications are killed before memory space is exhausted, making situations simpler. We expect that the findings in this article will be meaningful for emerging memory and storage management systems that make use of fast SCM media. In the future, we will perform the evaluation of our strategy with the complete system configurations consisting of the commercialized SCM products.

Conflict of Interest

The authors declare no conflict of interest.

Acknowledgment

This paper is an extension of work originally presented in the 6th IEEE Int'l Conf. on Information Science and Control Engineering [1]. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2019R1A2C1009275) and also by the ICT R&D program of MSIP/IITP (2019-0-00074, developing system software technologies for emerging new memory that adaptively learn workload characteristics).

References

[1] Y. Park, K. Cho, and H. Bahn, "Challenges and Implications of Memory Management Systems under Fast SCM Storage," in 2019 IEEE International Conference on Information Science and Control Engineering, 190-194, 2019. https://doi.org/10.1109/ICISCE48695.2019.00046
[2] S. Ng, "Advances in disk technology: performance issues," Computer, **31**(5), 75-81, 1998. http://doi.org/10.1109/2.675641
[3] C. Evans, "Flash vs 3D Xpoint vs storage-class memory: which ones go where?" Computer Weekly, **3**(9), 23-26, 2018.
[4] M. Stanisavljevic et al., "Demonstration of Reliable Triple-Level-Cell (TLC) Phase-Change Memory," in 2016 IEEE International Memory Workshop, 2016. https://doi.org/10.1109/IMW.2016.7495263
[5] S. Hyun, H. Bahn, and K. Koh, "LeCramFS: an efficient compressed file system for flash-based portable consumer devices," IEEE Trans. Consumer Electron., **53**(2), 481-488, 2007. https://doi.org/10.1109/TCE.2007.381719
[6] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," ACM SIGARCH Computer Architecture News, **37**(3), 24-33, 2009. https://doi.org/10.1145/1555815.1555760
[7] E. Lee, J. Jang, T. Kim, and H. Bahn, "On-demand snapshot: an efficient versioning file system for phase-change memory," IEEE Trans. Knowledge Data Engineering, **25**(12), 2841-2853, 2013. https://doi.org/10.1109/TKDE.2013.35
[8] R. K. Ramanujan, R. Agarwal, and G. J. Hinton, "Apparatus and Method for Implementing a Multi-level Memory Hierarchy Having Different Operating Modes," US Patent 20130268728 A1, Intel Corporation, 2013.
[9] B. Nale, R. Ramanujan, M. Swaminathan, and T. Thomas, "Memory channel that supports near memory and far memory access," PCT/US2011/054421, Intel Corporation, 2013.
[10] E. Lee, S. Yoo, and H. Bahn, "Design and implementation of a journaling file system for phase-change memory," IEEE Trans. Comput., **64**(5), 1349-1360, 2015. https://doi.org/10.1109/TC.2014.2329674
[11] S. Eilert, M. Leinwander, and G. Crisenza, "Phase change memory: A new memory enables new memory usage models," in 2009 IEEE International Memory Workshop, 2009.
[12] E. Lee, J. Kim, H. Bahn, S. Lee, and S. Noh, "Reducing Write Amplification of Flash Storage through Cooperative Data Management with NVM," ACM Transactions on Storage, **13**(2), 2017. https://doi.org/10.1145/3060146
[13] E. Lee, H. Bahn, S. Yoo, S. H. Noh, "Empirical study of NVM storage: an operating system's perspective and implications," in 2014 IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 405-410, 2014. https://doi.org/10.1109/MASCOTS.2014.56
[14] S. Lee, H. Bahn, and S. H. Noh, "CLOCK-DWF: a write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures," IEEE Trans. Comput., **63**(9), 2187-2200, 2014. https://doi.org/10.1109/TC.2013.98
[15] B. Anita, J. B. Chen, and N. P. Jouppi, "A simulation based study of TLB performance," in 1992 Int. Symp. Computer Architecture, 114-123, 1992.
[16] Valgrind, http://valgrind.org/
[17] N. Nethercote and J. Seward, "Valgrind: a program supervision framework," Electronic Notes in Theoretical Computer Science, **89**(2), 2003. https://doi.org/10.1016/S1571-0661(04)81042-9
[18] Freecell, https://en.wikipedia.org/wiki/FreeCell
[19] Gedit, https://wiki.gnome.org/Apps/Gedit
[20] Gqview, http://gqview.sourceforge.net
[21] Kghostview, https://linuxappfinder.com/package/kghostview
[22] Xmms, https://www.xmms.org/
[23] P. Weisberg and Y. Wiseman, "Using 4KB page size for virtual memory is obsolete," in 2009 IEEE International Conference on Information Reuse & Integration, 262-265, 2009. http://doi.org/10.1109/IRI.2009.5211562
[24] Facebook, https://www.facebook.com/
[25] Angrybirds, https://www.angrybirds.com/
[26] Youtube, https://www.youtube.com/
[27] Farmstory, https://play.google.com/store/apps/details?id=com.teamlava.farmstory
[28] Chrome, https://www.google.com/chrome/