IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 28, 2021, accepted March 24, 2021, date of publication March 31, 2021, date of current version April 12, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3069966

Modeling and Analysis of the Page Sizing Problem for NVM Storage in Virtualized Systems

YUNJOO PARK AND HYOKYUNG BAHN, (Member, IEEE)
Department of Computer Science and Engineering, Ewha Womans University, Seoul 120-750, Republic of Korea

Corresponding author: Hyokyung Bahn (bahn@ewha.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant by the Korean Government through MSIP under Grant 2019R1A2C1009275, and in part by the ICT Research and Development Program of MSIP/IITP (Developing system software technologies for emerging new memory that adaptively learn workload characteristics) under Grant 2019-0-00074.

ABSTRACT Recently, NVM (non-volatile memory) has advanced as a fast storage medium, and traditional memory management systems designed for HDD storage should be reconsidered. In this article, we revisit the page sizing problem in NVM storage, specially focusing on virtualized systems. The page sizing problem has not caught attention in traditional systems because of the two reasons. First, the memory performance is not sensitive to the page size when HDD is adopted as storage. We show that this is not the case in NVM storage by analyzing the TLB miss rate and the page fault rate, which have trade-off relations with respect to the page size. Second, changing the page size in traditional systems is not easy as it accompanies significant overhead. However, due to the widespread adoption of virtualized systems, the page sizing problem becomes feasible for virtual machines, which are generated for executing specific workloads with fixed hardware resources. In this article, we design a page size model that accurately estimates the TLB miss rate and the page fault rate for NVM storage. We then present a method that has the ability of estimating the memory access time as the page size is varied, which can guide a suitable page size for given environments. By considering workload characteristics with given memory and storage resources, we show that the memory performance of virtualized systems can be improved by 38.4% when our model is adopted.

INDEX TERMS Page size, NVM, virtualization, memory performance, address translation, page fault.

I. INTRODUCTION

For decades, the most commonly used page size in memory subsystems is 4KB. There have been attempts to change the page size [1]–[5], but 4KB is still commonly used. This is closely related to the characteristics of hard disk drive (HDD), which has been the main storage of computer systems. The access time of HDD is limited to tens of milliseconds, which is 5 to 6 orders of magnitude slower than DRAM access time [6]. Thus, the primary goal of memory subsystem design in traditional computer systems has been the minimization of storage accesses [7]. Also, as the seek movement is necessary for accessing data in HDD, which accounts for the major portion of storage access time, reading large data for each seek is efficient.

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

When considering these two characteristics of HDD (i.e., slow and seek), deciding a page size as large as possible would be a good solution. However, due to the limited capacity of main memory, reading large data from storage incurs the eviction of some other data in memory. Thus, increasing the size of data to be transferred is efficient only when the memory is sufficient or the data will be actually used. By considering this, read-ahead techniques are used to control the size of data to be transferred by monitoring the access characteristics of the data [8]. That is, if the access pattern is shown to be sequential, operating systems usually increase the number of pages to be read.

Recently, as the main memory capacity increases dramatically, there are attempts to use huge pages [9]–[11] and/or multiple size pages [12]–[17] in modern operating systems. However, the current operating systems like Linux just provide an option to change the page size of the system,

but they do not have the ability of adapting the page size to given system situations. Meanwhile, NVM (non-volatile memory) emerges as a new storage medium, and the page sizing problem is becoming an important issue. That is, our analysis shows that the memory system performance under NVM storage is sensitive to the page size, which was not the case for HDD storage systems.

Due to the recent advances in fast NVM storage technologies, the wide speed gap between memory and storage has been significantly reduced. The access time of NVM is about 1-100 times that of DRAM [18], [19], which is the significant improvement from the HDD case of 100,000 to 1,000,000 times. Another important change is that there is no seek movement in NVM so that it should pay much cost as the size of data to be transferred becomes large. That is, the gain of reading large data from each seek in HDD is not effective any longer in NVM. By considering this, using a small page can be an effective solution in NVM.

However, as storage accesses become very fast in NVM, the bottleneck of a memory access is shifting to the address translation between logical and physical addresses. Note that the memory access time consists of the address translation time and the data access time. As the data access time becomes very small by the reduced page fault handling latency in NVM storage, improving the TLB miss rate during address translation matters significantly. Although a small page can be adopted in order to improve the data access time, it increases the TLB miss rate, which eventually degrades the address translation time. To improve the address translation time, the page size should be large. This is because the TLB miss rate would be improved when the limited number of TLB entries covers more memory area, and it can be realized by increasing the page size.

Because of the two conflicting reasons aforementioned, it is necessary to select the page size by considering the relative impact of the address translation and the data access in given situations. Also, depending on whether the memory capacity is sufficient or not, an appropriate page size may change, which should also be considered in deciding the page size, making the problem even more complicated.

To cope with this situation, we design a page size model that accurately estimates the TLB miss rate and the page fault rate as the page size is varied. We then present a method that has the ability of estimating the memory access time for various page sizes, which can guide a suitable page size for given system environments.

Meanwhile, page sizing problems have not caught attention in traditional systems because changing the page size of a system incurs significant performance overhead during the page size transition, not being an easy matter in real system situations. However, due to the widespread adoption of virtualization techniques in desktop PCs as well as cloud systems, selecting the page size for each virtual machine becomes a feasible solution. This is because virtual machines are usually created for executing specific workloads, and the resource configurations for each virtual machine are decided while it is generated.

For example, the memory capacity, the storage type, and the workload type are determined for each virtual machine as shown in Fig. 1, and it is possible to find an appropriate page size for the given virtual machine when it starts. That is, the optimized page size depends on whether the main memory size is sufficient or not, the storage is HDD or NVM, and the workload is memory-intensive or computing-intensive, and selecting the page size is possible by considering the characteristics of the virtual machine. We present the page size model for NVM, and show the effectiveness of this model on virtualized systems. By considering workload characteristics with given memory and storage resources, we show that the memory system performance can be improved by 38.4% on average and up to 55.7% when the page size of a virtual machine is determined based on our model.

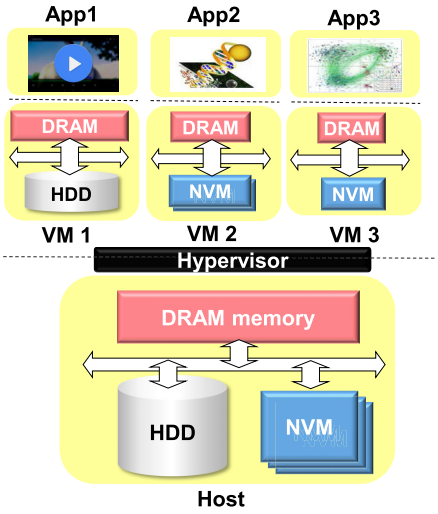


FIGURE 1. A host system with HDD and NVM storage and the three virtual machines on top of it with different resource configurations and workload characteristics.

The remainder of this article is organized as follows. Section II briefly explains the motivation of this research. Section III presents the analysis and modeling of memory access time, specially focusing on the page fault rate and the TLB miss rate. In Section IV, we conduct the validation of our model with respect to the accuracy and the effectiveness in virtualized systems. Section V briefly summarizes the studies related to this article. Finally, we conclude this article in Section VI.

II. MOTIVATIONS

In this section, we overview the memory system performances when NVM-based storage is adopted. Specifically, we observe the influence of the page size as the storage media change. We compare the memory access time when using HDD and NVM as the underlying storage device in order to see the impact of the page size on memory system performances. As HDD and NVM have two different

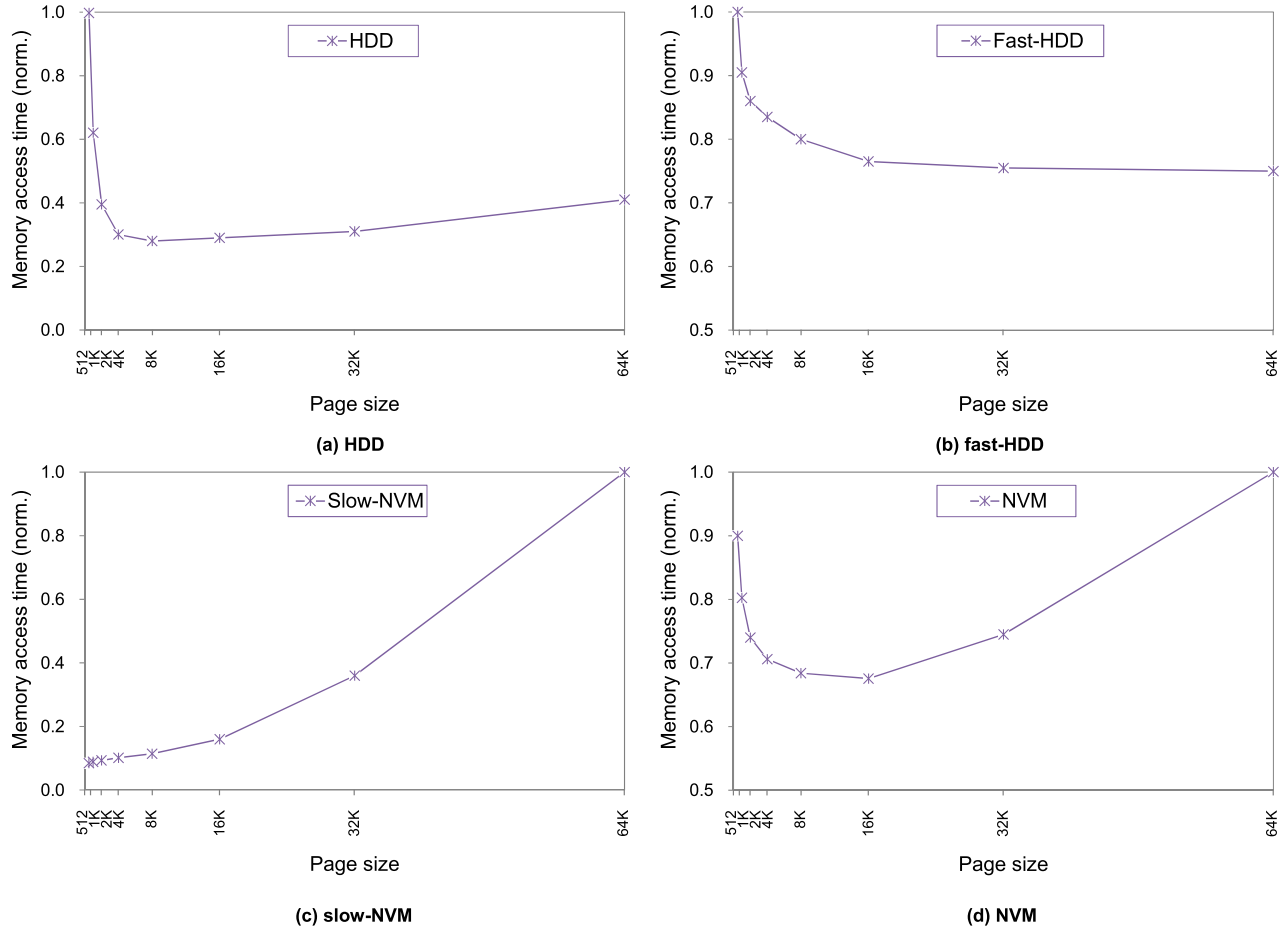


FIGURE 2. Memory access time under different types of storage systems.

aspects, i.e. access speed and existence of seek movement, we additionally simulate two hypothetical storage media: slow-NVM, which is as slow as HDD but does not have seek movement similar to NVM, and fast-HDD, which is as fast as NVM but has seek movement of head like HDD.

Fig. 2 shows the normalized memory access time of a Linux system as the page size is varied for HDD, NVM, fast-HDD, and slow-NVM, respectively. (The characteristics of the workloads we experimented will be explained later in Section III.) As shown in the figure, HDD storage shows good performances if the page size is 4KB or more, but the performance is not sensitive when the page size is larger than 4KB. In HDD-based systems, there are 5 to 6 orders of magnitude speed gap between memory and storage, so the page fault rate is the primary factor that accounts for the performance of the system. Thus, memory systems should be designed to minimize the page fault rate.

In the case of NVM, as shown in Fig. 2(d), neither a large page nor a small page performs well, and the memory system performance is sensitive to the page size. This is because the memory access time is influenced by the two conflicting factors: the TLB miss rate and the page fault rate. Each factor affects the address translation time and the data access time, respectively, which are the two time components of the memory access time. Thus, we need to consider the impact of these two factors in the given environment, and judiciously determine an appropriate page size.

One interesting result is that the page sizing problem is not complicated in fast-HDD and slow-NVM as shown in Figs. 2(b) and 2(c), respectively. That is, a large page performs well in fast-HDD as it improves the address translation time by reducing the TLB miss rate and also improves the data access time by considering the large seek cost for each storage access. In case of slow-NVM, a small page performs well as storage access is the crucial factor that determines memory performances, but the cost of storage access is proportional to the size of data. Thus, it is not necessary to transfer large data unless they are known to be actually used.

Fig. 3 separately shows the address translation time and the data access time for HDD and NVM storage systems. As shown in the figure, the data access time accounts for the most of the memory access time in the HDD-based system. The address translation time is negligible, and thus the goal of the memory system design focuses only on the minimization of storage accesses. On the other hand, as shown in Fig. 3(b),

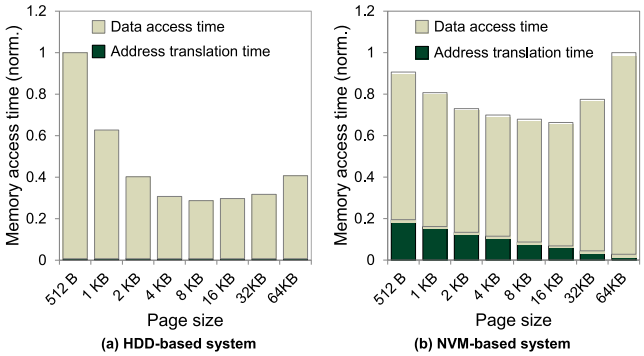


FIGURE 3. Memory access time as a function of the page size.

the impact of the address translation time is not negligible in the NVM-based system. A large page size reduces the address translation time by improving the TLB miss rate. In contrast, a small page size reduces the data access time by decreasing the page fault handling time, which depends on the size of data to be transferred. Thus, the page size should be selected by considering these trade-off relations. This is not an easy matter as the effect of the two time components depends not only on the memory capacity but also on the system situations. To accurately estimate the relation between the memory access time and the page size, we will present a model that fits the address translation time and the data access time well by making use of appropriate functions.

Before concluding this section, let us briefly discuss the case of flash-based SSD (solid state drives). SSD is widely adopted in various kinds of systems ranging from mobile laptops to cloud servers. SSD has no seek like NVM, but the access latency is about 3 orders of magnitude slower than NVM. We performed experiments under such configurations and observed that the result for SSD is similar to that of slow-NVM rather than NVM.

That is, SSD is just 10 to 100 times faster than HDD, but NVM is over 10,000 times and up to 100,000 times faster than HDD. Thus, SSD and NVM are very different types of storage, and in the case of SSD, determining the page size is not a complicated problem similar to slow-NVM. This is because the impact of the address translation time is not significant, and thus the trade-off relationship between the TLB miss rate and the page fault rate does not form in SSD. This does not simply indicate that the page sizing in SSD is unnecessary, but we are not interested in this as the problem is not challenging in academic perspectives. Nevertheless, as the performance of SSD is being improved, the problem will be similar to that of NVM storage, and thus the page sizing will be increasingly sensitive to various situations including the address translation time, the memory size, and workload characteristics. Thus, it will be a topic of our future research if the performance of SSD becomes close to that of NVM.

III. ANALYSIS AND MODELING OF MEMORY ACCESS TIME

Modern general-purpose operating systems manage the main memory by the fixed size unit called *page*. The most common page size is 4KB, which is also the basic unit for transferring memory data to storage. However, modern operating systems like Linux also support the read-ahead (or prefetching) function to load up to 128 adjacent pages from storage if a page fault happens. The rationale behind this lies in the characteristics of HDD, which requires high seek cost per each storage access irrespective of the data size to be transferred. That is, loading large data for each storage access will be efficient in HDD storage. In some large memory systems, huge pages of up to 4MB are also supported for HDD-based systems. However, this will not be efficient for NVM storage as it is fast and does not have seek movement.

Meanwhile, decreasing the page size is not an easy matter as it will do harm to the address translation process. That is, a fixed number of TLB entries can cover more memory address space if the page size grows, which can improve the address translation time. Thus, the page size for NVM-based systems should be carefully selected by considering these overall situations.

To see the effect of the page size on the memory access time, we analyze and model the TLB miss rate and the page fault rate as the page size is varied. Then, we verify our model by using real memory access traces captured from various Linux applications. The characteristics of these traces are described in Table 1.

We opened the memory access trace of the five applications in our Github page [20]. The game trace was collected while playing the traditional card game application called Freecell; the office trace was extracted while editing a document file by the text editor software gedit; the photo trace was collected during the execution of the image view and organizer application Geeqie; the PDF trace was captured while viewing a PDF file by the document viewer application KGhostview; and the multimedia trace was collected while playing a media file through the Linux media player.

As our traces were captured at the instruction level memory references, they have complicated nested loops, which are difficult to be described through a single characteristic like file access patterns [21]. Thus, we focus on the characteristics of the workloads with respect to the locality of references. Specifically, we analyze the access density of the references based on the footprint of the applications and the total number of memory accesses that occur within the footprint. As listed in Table 1, the access density of the game application is the lowest among the five applications we experimented, which implies that a relatively low performance can be expected when the same ratio of resources (i.e., TLB size or memory size) per footprint is provided. On the contrary, the access densities of the multimedia and office applications exhibit very high, and thus, we can expect relatively high

TABLE 1. Workload characteristics.

Workload	Footprint (MB)	Access density (# of accesses per 4KB)	Memory access count		
			Write	Read	Total
Game	9.84	194.59	60,040	430,135	490,175
Office	14.12	479.64	132,822	1,600,941	1,733,763
Photo	7.26	328.58	345,399	365,286	610,685
PDF	16.98	355.69	103,540	1,442,595	1,546,135
Multimedia	7.86	580.94	978,242	190,697	1,168,939

performances when the same proportion of the resources are provided.

A. MEMORY ACCESS TIME

As the main memory of modern computer systems is usually managed based on virtual memory paging, translation of logical to physical addresses is necessary in order to access a memory page. This is conducted by accessing the page table, which is resident in main memory. To accelerate the address translation process, a certain subset of the page table is maintained in the Translation Look-aside Buffer (TLB), which behaves as a hardware cache of the page table.

Accordingly, in order to access a memory page, address translation by TLB is firstly tried. The page table is, then, accessed only if the address translation by TLB fails. Then, the actual data is accessed in the memory location pointed by the translated address. During this process, if the requested data does not exist in main memory, it should be loaded from storage, which we call the page fault. Assume that the TLB miss rate is r, and the page fault rate is f. Then the memory access time T_{MEM} is represented as

$$T_{MEM} = T_{ADDR} + T_{PAGE} \tag{1}$$
$$T_{ADDR} = (1-r)^{*}t_{\varepsilon} + r^{*}(t_{\varepsilon} + t_{\tau}) \tag{2}$$
$$T_{PAGE} = (1-f)^{*}t_{\tau} + f^{*}(t_{\tau} + T_{PF}) \tag{3}$$

where T_{ADDR} is the address translation time, T_{PAGE} is the data access time, t_{ε} is the latency to access TLB entries, t_{τ} is the latency to access memory, and T_{PF} is the page fault handling time.

As the TLB capacity is limited, inserting a new entry needs the eviction of an existing entry if no free entry is available. In order to decide the entry to be evicted, we use the LRU (Least Recently Used) replacement algorithm, which is commonly used in associative caches including TLB. Since the number of pages in main memory is also fixed, we need another replacement algorithm. In particular, if a page should be loaded from storage but no free page is available in memory, we should evict an existing page from memory. For selecting a victim page in memory, we use the CLOCK replacement algorithm as it is a representative algorithm used in paging systems [22]. Note that Linux also adopts a variant of the CLOCK algorithm by making use of two page lists, the active list and the inactive list, and replaces pages not used recently in the inactive list.

B. TLB MISS RATE

With the limited number of TLB entries, a large page can cover more memory address space, probably improving the TLB miss rate. To validate this in real situations, we perform experiments for various workloads and page sizes. Fig. 4 shows this result. As can be seen from this figure, the TLB miss rate is improved as the page size grows from 512B to 64KB for all workloads we experimented.

To predict the TLB miss rate for a given system or workload precisely, we perform the fitting of the TLB miss rate as a function of the page size. In particular, we use the three simple and typical fit functions and compare the adjusted R^2 and RMSE values. The three fit functions are as follows:

$$f_1(x) = ax^2 + bx + c \tag{4}$$
$$f_2(x) = ax^b \tag{5}$$
$$f_3(x) = ae^{bx} \tag{6}$$

where (4), (5), and (6), respectively, represent the quadratic-fit, power-fit, and exponential-fit.

Fig. 5 shows some examples of the fit with the actual data. As can be seen from the figure, the power fit models the TLB miss rate the best. For more accurate validation, we calculate the adjusted R^2 and RMSE values of the three fits as shown in Tables 2 and 3. As listed in the table, the power-fit results in the highest adjusted R^2 of 0.947 as well as the lowest RMSE of 0.033 on average.

C. PAGE FAULT RATE

The page fault rate depends on the available memory size as well as the page size. Since the main memory capacity is fixed, the number of page frames decreases as the page size grows. The page fault rate follows the law of the diminishing marginal utility as the page size increases. That is, the page fault rate curve has an inflection point. With the given memory capacity, the page fault rate is improved as the page size grows due to the principle of locality. However, after that point, the page fault rate is degraded again because the limited memory capacity cannot accommodate a variety of data. This point, of course, is varied based on the characteristics of workloads. In particular, if the memory size is not sufficient,

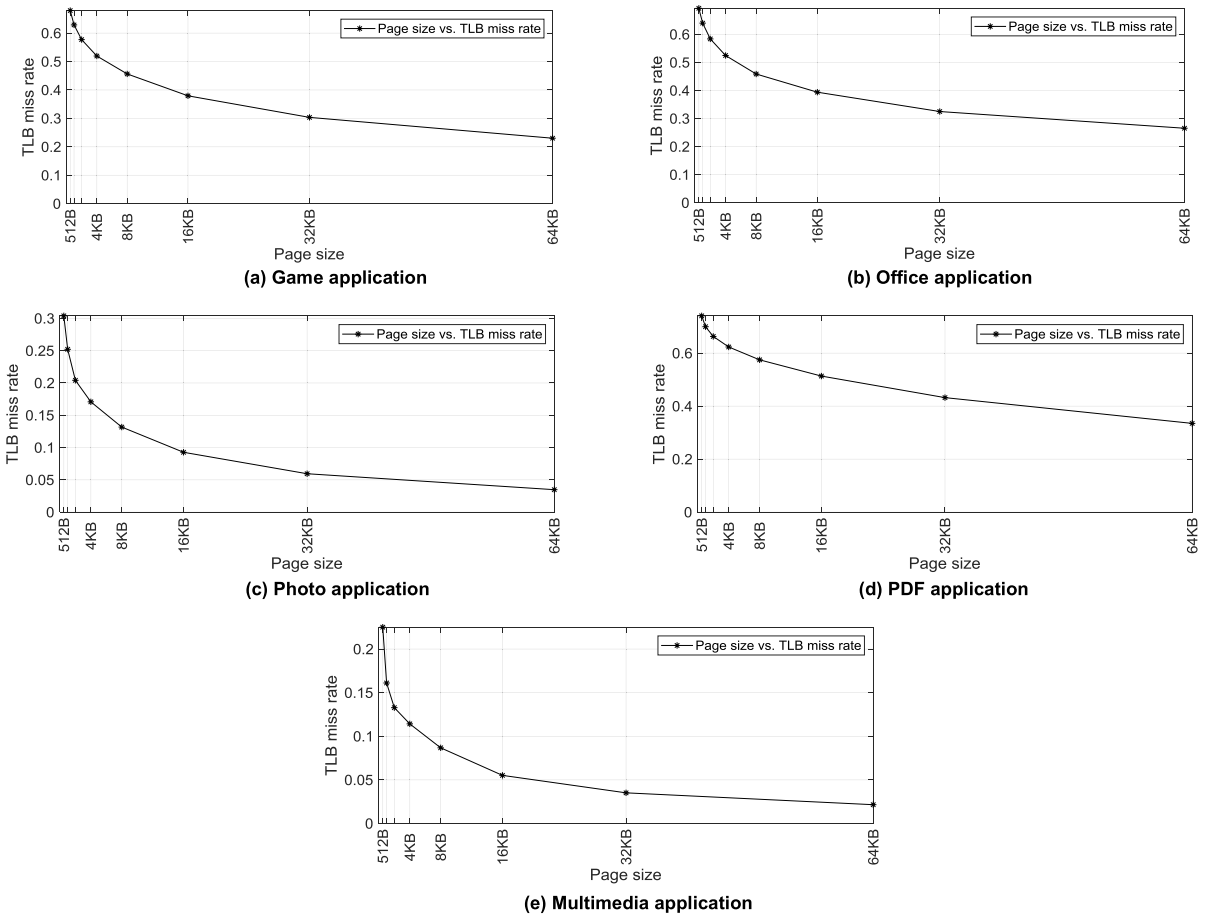


FIGURE 4. The TLB miss rate as a function of the page size.

TABLE 2. The adjusted R^2 value of the TLB miss rate as the fit functions are varied.

Workload	Adjusted R^2		
	Quadratic (f_1)	Exponential (f_2)	Power (f_3)
Game	0.876	0.858	0.938
Office	0.841	0.794	0.965
Photo	0.734	0.906	0.968
PDF	0.941	0.912	0.888
Multimedia	0.663	0.878	0.976
Average	0.811	0.869	0.947

TABLE 3. RMSE of the TLB miss rate as the fit functions are varied.

Workload	RMSE		
	Quadratic (f_1)	Exponential (f_2)	Power (f_3)
Game	0.064	0.068	0.045
Office	0.068	0.077	0.032
Photo	0.051	0.031	0.018
PDF	0.042	0.051	0.057
Multimedia	0.041	0.025	0.011
Average	0.053	0.050	0.033

the page fault rate increases more rapidly after the inflection point.

Fig. 6 shows the page fault rate curves as a function of the page size with the memory capacity of 30% to 100%. The memory capacity of 100% implies the configuration that the complete footprint of the workload can be loaded into memory. As we see from the figure, except for the photo application, the page fault rate increases after a certain page size. For the photo application, we can also see this trend when the page size is even larger than the ranges we experimented. To find the function that fits this trend, we use an exponential fit composed of two terms as follows.

$$f(x) = ae^{bx} + ce^{dx} \tag{7}$$

As shown in Fig. 7, the exponential-fit with two terms works well in most cases, but it seems that the accuracy is degraded when the memory size is relatively small. The adjusted R^2 and RMSE are compared under different memory

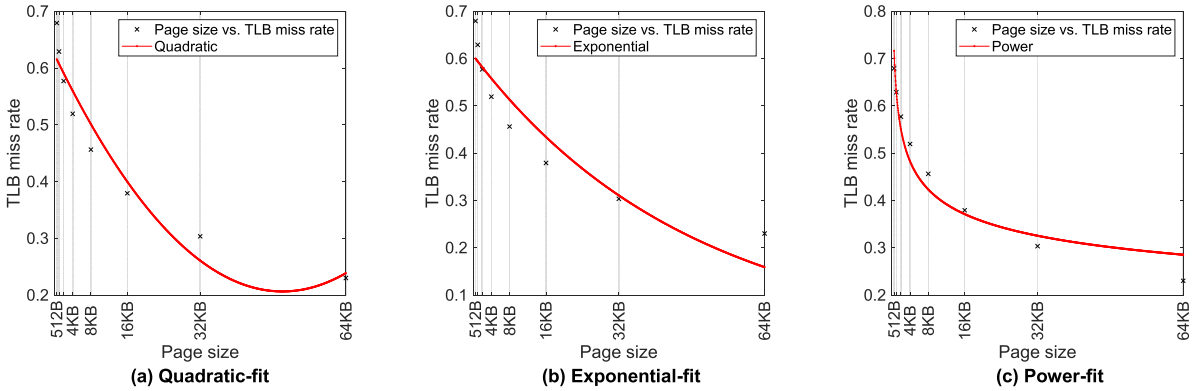


FIGURE 5. Examples of the fit functions.

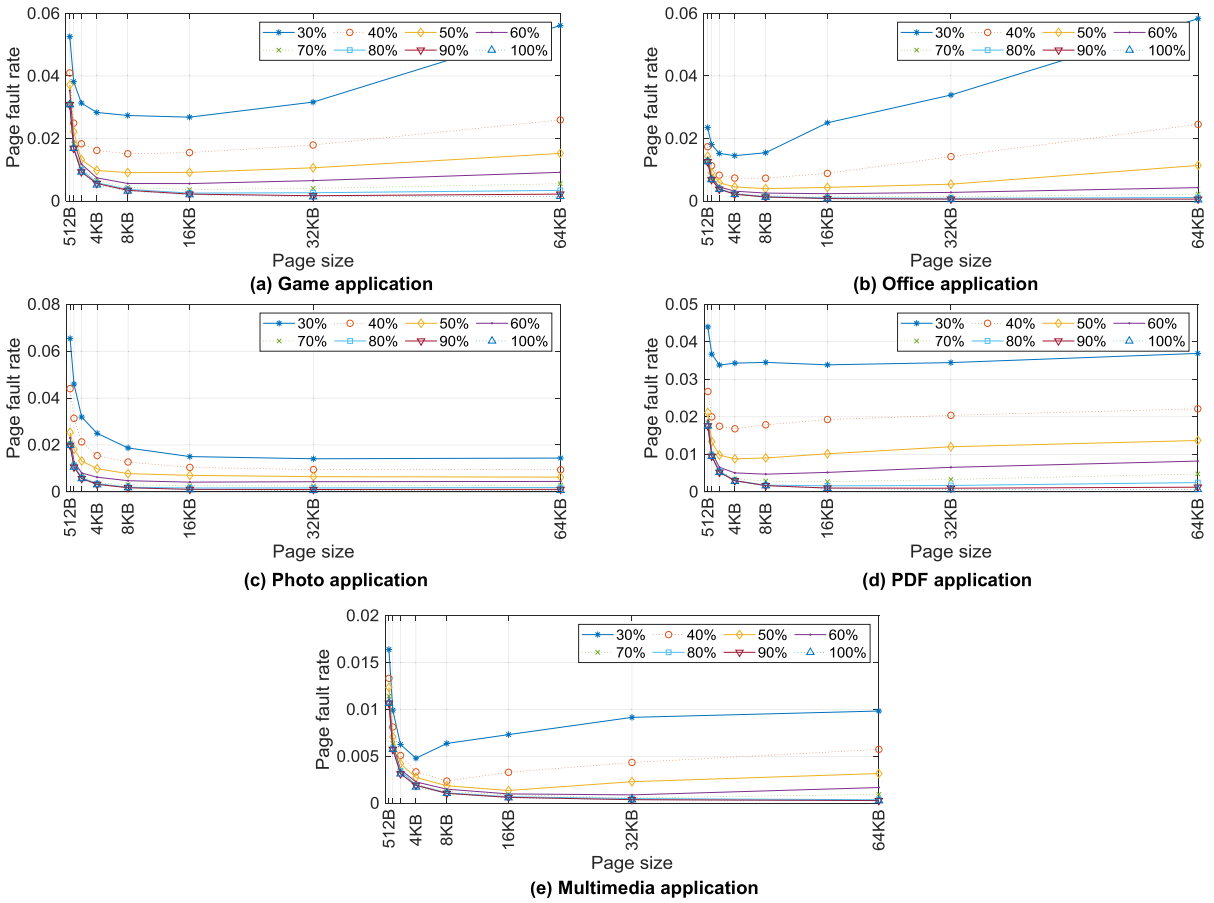


FIGURE 6. The simulated and modeled page fault rate as a function of the page size.

sizes to verify the suitability of our exponential-fit. As shown in Tables 4 and 5, the modeled page fault rate tends not to fit at small memory sizes. Therefore, we use a paired t-test to validate whether the measured and the modeled page fault rates are statistically different. The null hypothesis of the test states that the difference between the measured and the modeled page fault rates is equal to zero. Based on the t-statistics in Table 6, the tests fail to reject the null hypothesis for all five tested workloads. In other words, there is no statistically significant evidence to conclude that the measured and the modeled page fault rates are different. Thus, we apply the exponential-fit to the page fault rate model.

IV. VALIDATIONS AND IMPLICATIONS

This section validates the model we designed for the page fault rate and the TLB miss rate with respect to the memory

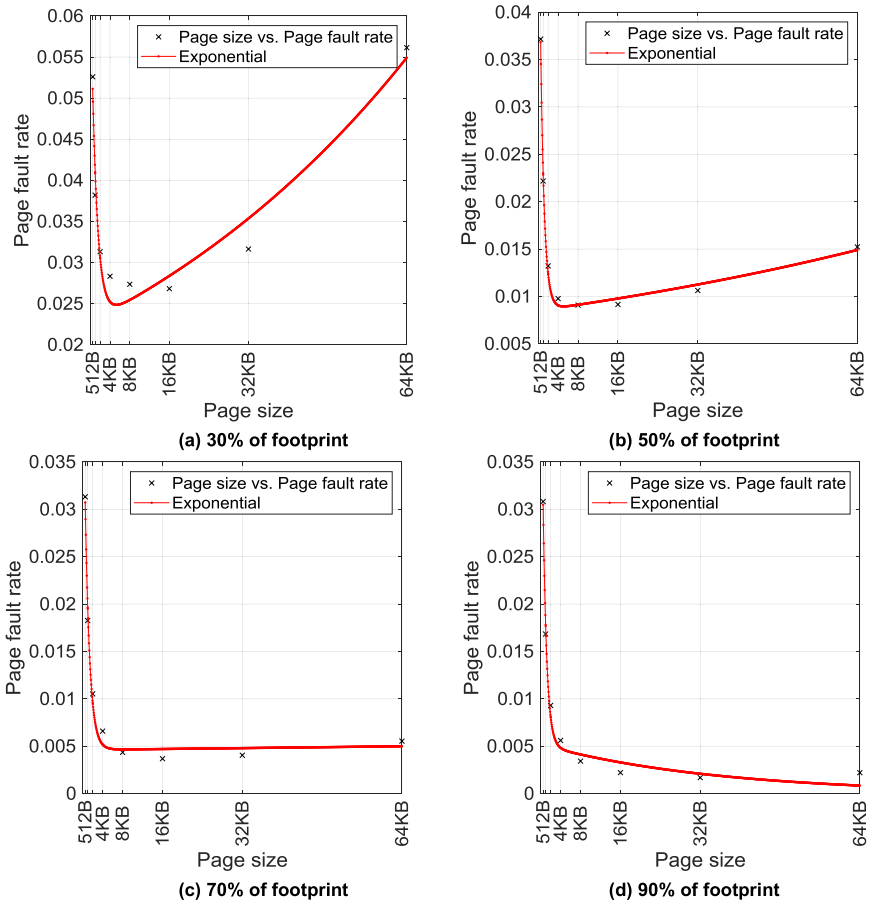


FIGURE 7. The simulated and modeled page fault rate as a function of the page size.

access time, and then assesses the effectiveness of this model. Specifically, the evaluation is conducted under a variety of virtual machine environments.

Fig. 8 shows the comparisons of the results from our model and those from the replaying experiment under the 30% memory capacity. As shown in the figure, our model predicts the memory access time well. Although the predicted memory access time by our model does not find the exact value, the trend is very similar. Note that it is more important to predict trends rather than exact values when determining the page size to minimize the overall memory access time.

To see the similarity of the two curves, we calculate the Pearson correlation coefficient as shown in Table 7. Note that the value is close to 1 when the two curves are similar. As listed in Table 7, the Pearson coefficient is more than 0.99 in all cases, implying that our model predicts the memory access time accurately for given page sizes.

As the proposed model has the ability of predicting memory access time well, we can make use of the model in determining the page size for the given system accordingly. To assess the effectiveness of our model, we perform simulation experiments under the 5 scenarios consisting of 12 virtual machines. Table 8 lists the configurations of each scenario we experimented.

Scenario 1 consists of three virtual machines, VM-1, VM-2, and VM-3, which execute game, office, and photo applications, respectively, with the memory size of 80%, 10%, and 60%, and the storage of NVM. Similarly, Scenario 2 consists of three virtual machines, VM-4, VM-5, and VM-6, which execute PDF, multimedia, and game applications, respectively, with the memory size of 90%, 50%, and 30%. VM-4 and VM-5 use NVM as storage, whereas VM-6 uses HDD. Scenario 3 consists of two virtual machines, VM-7 and VM-8, which perform the office and photo applications, respectively, with the same resource configurations of NVM storage and 80% memory. Scenario 4 consists of two virtual machines, VM-9 and VM-10, which execute the PDF and multimedia applications, respectively, with the memory size of 20% and 80%, and the storage of NVM. Scenario 5 consists of two virtual machines, VM-11 and VM-12, which execute the game and office applications, respectively, with the memory size of 60% and 40%, and the storage of NVM and HDD.

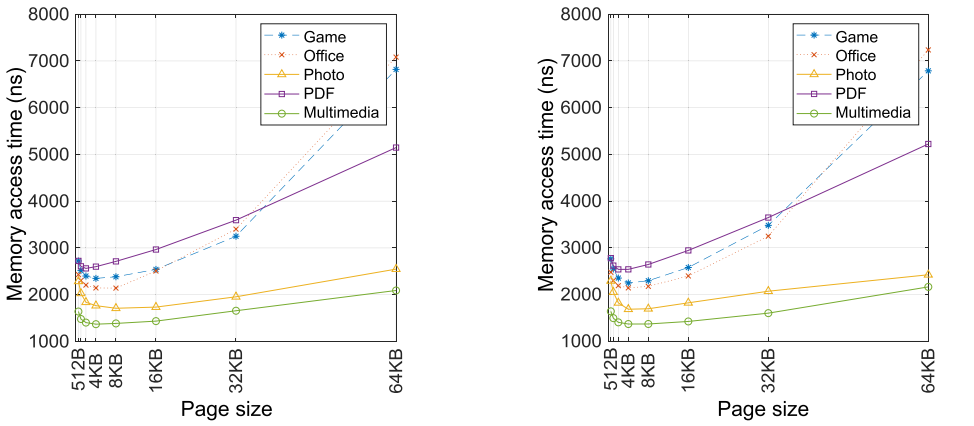
Fig. 9 shows the memory access time of the system that uses the page size based on our model in comparison with the original system with the 4KB page. We also simulated an adaptive page sizing approach based on the Hill climbing method and compared the result with our model. Note that

TABLE 4. The adjusted R^2 value of the page fault rate as the fit functions are varied.

Workload	Adjusted R^2							
	30%	40%	50%	60%	70%	80%	90%	100%
Game	0.9230	0.9744	0.9924	0.9777	0.9811	0.9755	0.9838	0.9900
Office	0.9544	0.9829	0.9724	0.9741	0.9784	0.8630	0.9891	0.9936
Photo	0.9752	0.9881	0.9851	0.9799	0.9883	0.9874	0.9851	0.9922
PDF	0.9727	0.9548	0.9822	0.9908	0.9874	0.9796	0.9813	0.9924
Multimedia	0.9193	0.9780	0.9694	0.9686	0.8854	0.9868	0.9944	0.9947
Average	0.9489	0.9756	0.9803	0.9782	0.9641	0.9585	0.9867	0.9926

TABLE 5. RMSE of the page fault rate as the fit functions are varied.

Workload	RMSE							
	30%	40%	50%	60%	70%	80%	90%	100%
Game	0.0032	0.0014	0.0008	0.0015	0.0013	0.0016	0.0013	0.0010
Office	0.0032	0.0008	0.0006	0.0006	0.0006	0.0015	0.0004	0.0003
Photo	0.0029	0.0014	0.0008	0.0009	0.0007	0.0007	0.0008	0.0006
PDF	0.0006	0.0007	0.0005	0.0005	0.0006	0.0008	0.0008	0.0005
Multimedia	0.0010	0.0005	0.0006	0.0006	0.0013	0.0004	0.0003	0.0003
Average	0.0022	0.0010	0.0007	0.0008	0.0009	0.0010	0.0007	0.0005



(a) Simulated memory access time by replaying experiments. (b) Predicted memory access time by our model.

FIGURE 8. Validation of the proposed model.

TABLE 6. The T-Test between the modeled and the actual data.

Workload	Sample size	*t*-value	*p*-value
Game	64	-0.280	0.780
Office	64	-0.844	0.402
Photo	64	-0.105	0.917
PDF	64	-0.357	0.722
Multimedia	64	-1.449	0.152

Hill climbing is an empirical method that attempts to find a better solution by making an incremental change to the solution until no further improvements can be found.

As shown in the figure, the proposed model improves the memory access time significantly compared to the original system. Specifically, the improvement is 38.4% on average. The largest improvement is observed in VM-4, where the system adopting our model shows 55.7% better than the original system. This is because the memory size of VM-4 is relatively large, and thus 4KB is far from the page size suitable for this situation. Note that the proposed model shows even better results as the memory size increases by improving the address translation time. Nevertheless, our model improves the memory access time by 19.0% in VM-2, although it has a relatively small memory size. This is due to the characteristics of the application executed in VM-2. That is, the office

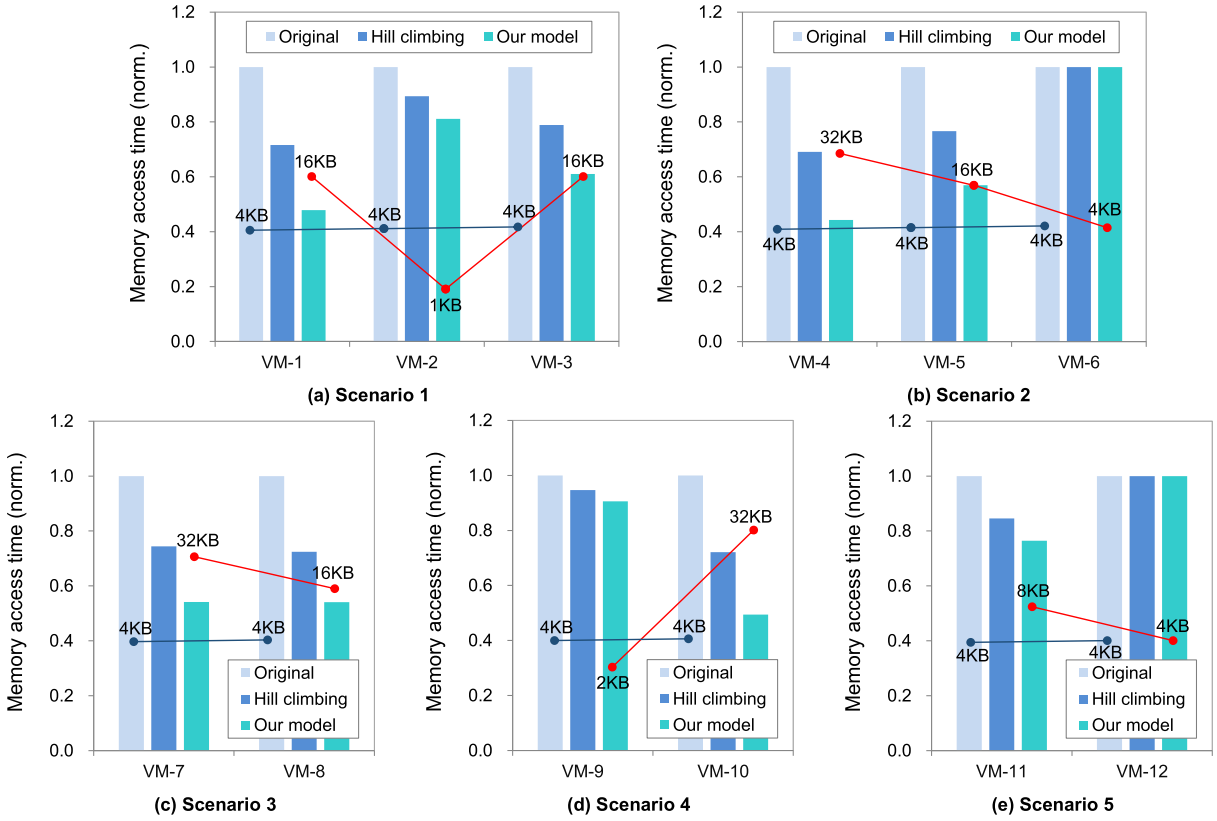


FIGURE 9. Comparisons of the original 4KB page configuration and those with the proposed model and the Hill climbing method.

TABLE 7. Pearson correlation coefficients between the modeled and the actual memory access time.

Workload	Pearson (ρ)
Game	0.997
Office	0.998
Photo	0.990
PDF	0.996
Multimedia	0.992
Average	0.995

application we executed in VM-2 is memory-intensive, and thus the memory performance exhibits sensitive to the page size. There is no performance improvement in VM-6 and VM-12 as they adopt HDD storage, and thus our model was not adopted. Except for these two cases, NVM storage is used and our model improves the memory performance in all cases ranging from 10.0% to 55.7% according to the workload and system situations.

Now, let us see the page size chosen by our model. As shown in Fig. 9, the selected page size is not identical, but ranges widely from 1KB to 32KB according to the relative memory size and the workload characteristics. Specifically, the page size was 1KB in VM-2, 2KB in VM-9, 4KB in VM-6 and VM-12, 8KB in VM-11, 16KB in VM-1, VM-3, VM-5, and VM-8, and 32KB in VM-4, VM-7, and VM-10. When the memory size of the VM is relatively small or the locality of the workload is weak, page faults account for a large portion of the memory access time, and thus a relatively small page performs well. VM-2 and VM-9 are such cases. In contrast, when the memory size is sufficient or the locality of the workload is strong, the address translation process accounts for a large portion in memory performance, and a relatively large page performs well. VM-1, VM-3, VM-4, VM-5, VM-7, VM-8, and VM-10 are such cases.

Now, let us discuss the performance of our model in comparison with the Hill climbing method. As shown in Fig. 9, Hill climbing also yields better performance than the original system with the 4KB page, but the performance improvement of our model against Hill climbing is 22.5% on average. This is because our model estimates a suitable page size for the given VM environment precisely, whereas Hill climbing finds an appropriate page size through trial and error. Note also that Hill climbing needs the overhead of transition as the page size is gradually changed.

V. RELATED WORK

Using NVM in various storage layers has been attempted. NVM has both memory and storage features but its physical characteristics are different to DRAM memory or

TABLE 8. Experimental scenarios used in the experiments.

Scenario	VM	Workload	Resource configurations	Page size chosen
Scenario 1	VM–1	Game	Memory size (80% of footprint), storage type (NVM)	16KB
	VM–2	Office	Memory size (10% of footprint), storage type (NVM)	1KB
	VM–3	Photo	Memory size (60% of footprint), storage type (NVM)	16KB
Scenario 2	VM–4	PDF	Memory size (90% of footprint), storage type (NVM)	32KB
	VM–5	Multimedia	Memory size (50% of footprint), storage type (NVM)	16KB
	VM–6	Game	Memory size (30% of footprint), storage type (HDD)	4KB
Scenario 3	VM–7	Office	Memory size (80% of footprint), storage type (NVM)	32KB
	VM–8	Photo	Memory size (80% of footprint), storage type (NVM)	16KB
Scenario 4	VM–9	PDF	Memory size (20% of footprint), storage type (NVM)	2KB
	VM–10	Multimedia	Memory size (80% of footprint), storage type (NVM)	32KB
Scenario 5	VM–11	Game	Memory size (60% of footprint), storage type (NVM)	8KB
	VM–12	Office	Memory size (40% of footprint), storage type (HDD)	4KB

HDD storage media. Thus, a number of studies have been conducted on the efficient management of NVM when it is adopted in various storage hierarchies of computer systems.

In early work, as the size of NVM was not large enough for main storage, a certain limited part of the total storage image could be located on NVM. Specifically, important data (e.g., metadata) or hot data can be stored on NVM. The Protected and persistent RAM File System (PRAMFS) is devised to maintain frequently accessed data in NVM for fast reboot and power cycle intact without flushing them to slow HDD [23]. Edel *et al.* propose MRAMFS, which maintains metadata on NVM [24]. In order to efficiently manage the limited NVM storage space, they store metadata in a compressed form.

As the capacity of NVM grows significantly, studies on NVM storage have focused on the file system design that retains the total file system image on NVM. Condit *et al.* propose BPFS, which is a new copy-on-write file system for NVM that makes use of the byte-addressable feature of NVM [25]. In particular, BPFS updates data in-place if the modifications are smaller than an atomic operation unit. This can eliminate a large proportion of out-of-place updates that occur in copy-on-write file systems. Lee *et al.* propose another copy-on-write file system for NVM called OND [26]. Unlike conventional copy-on-write file systems that need the propagation of out-of-place updates up to the file system root, OND breaks the recursive writes at the immediate parent level.

Wu and Reddy present an in-memory file system for NVM called SCMFS (Storage-Class Memory File System) [27]. SCMFS allows file accesses via memory interfaces, thereby eliminating the overhead of software stacks in file I/Os. Lee *et al.* propose a new journaling file system for NVM called Shortcut-JFS [18]. Unlike traditional journaling file systems that require the writing of an entire block twice for journaling and checkpointing, Shortcut-JFS reduces the writing by more than half based on the differential logging and in-place checkpointing techniques.

There have been studies on efficient user-defined storage interfaces utilizing the byte-addressability of NVM. Coburn *et al.* present NVM-based in-memory persistent data store, which allows users to generate a persistent data structure based on NVM [28]. Volos *et al.* suggest a programming interface for NVM called Mnemosyne that allows the creation and management of data in NVM without the risk of inconsistency [29]. In particular, Mnemosyne enables users to declare persistent data with the given primitives, which ensure transaction management, thereby guaranteeing data consistency against system failures. Peng *et al.* present a user-space page management scheme that exploits application and storage characteristics [30]. Specifically, they use application hints to improve the selection of caching, prefetching, and eviction policies. They also support various page sizes appropriately for each user memory region, balancing the overhead and data usage.

Recently, studies on NVM storage focus on the simplification of traditional software stacks and I/O paths designed for slow HDD storage. Yang *et al.* suggest synchronous I/O for fast NVM storage by showing its better performance than interrupt-based asynchronous I/O if storage is sufficiently fast [31]. Caulfield *et al.* present an efficient software and hardware interfaces for fast NVM storage based on the quantification of each software stack's overhead during storage I/Os [32]. Lee *et al.* investigate the NVM storage performance for a wide range of operating system configurations, and show that the effect of synchronous I/O, direct I/O, and read-ahead in NVM is different to that in HDD storage [19].

VI. CONCLUSION

In this article, we revisited the page sizing problem in NVM storage, specially focusing on virtualized systems. Unlike traditional systems where changing the page size is not feasible, we showed that the page sizing problem has come into a new situation due to the widespread adoption of virtualized systems and the emergence of fast NVM storage. This article analyzed the memory access time in NVM-based storage separately for the address translation time and the data access time as the page size is varied. We then designed a page size model that accurately estimates the TLB miss rate and the page fault rate for NVM storage. By using this model,

it is possible to estimate the effect of the page size on the memory system performances, and we can guide an appropriate page size for given system environments. We showed that the memory system performance of virtualized systems can be improved by 38.4% on average and up to 55.7% by considering workload characteristics and given resource configurations in deciding the page size of a virtual machine based on our model.

REFERENCES

[1] Y. Park and H. Bahn, "Management of virtual memory systems under high performance PCM-based swap devices," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, Jul. 2015, pp. 764–772.
[2] P. Weisberg and Y. Wiseman, "Using 4KB page size for virtual memory is obsolete," in *Proc. IEEE Int. Conf. Inf. Reuse Integr.*, Aug. 2009, pp. 262–265.
[3] Y. Du, M. Zhou, B. R. Childers, D. Mossé, and R. Melhem, "Supporting superpages in non-contiguous physical memory," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 223–234.
[4] J. Navarro, S. Iyer, P. Druschel, and A. Cox, "Practical, transparent operating system support for superpages," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 89–104, Dec. 2002.
[5] Y. Park and H. Bahn, "Challenges in memory subsystem design for future smartphone systems," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Feb. 2017, pp. 255–260.
[6] E. Lee and H. Bahn, "Caching strategies for high-performance storage media," *ACM Trans. Storage*, vol. 10, no. 3, pp. 1–22, Jul. 2014.
[7] S. W. Ng, "Advances in disk technology: Performance issues," *Computer*, vol. 31, no. 5, pp. 75–81, May 1998.
[8] A. Laga, J. Boukhobza, M. Koskas, and F. Singhoff, "Lynx: A learning linux prefetching mechanism for SSD performance model," in *Proc. 5th Non-Volatile Memory Syst. Appl. Symp. (NVMSA)*, Aug. 2016, pp. 1–6.
[9] Y. Kwon, H. Yu, S. Peter, C. J. Rossbach, and E. Witchel, "Coordinated and efficient huge page management with Ingens," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 705–721.
[10] N. Agarwal and T. Wenisch, "Thermostat: Application-transparent page management for two-tiered main memory," in *Proc. 22th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2017, pp. 631–644.
[11] X. Wang, H. Liu, X. Liao, J. Chen, H. Jin, Y. Zhang, L. Zheng, B. He, and S. Jiang, "Supporting superpages and lightweight page migration in hybrid memory systems," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 2, pp. 1–26, Jun. 2019.
[12] N. Ganapathy and C. Schimmel, "General purpose operating system support for multiple page sizes," in *Proc. Annu. Tech. Conf. (ATC)*, 1998, pp. 91–104.
[13] *Sun BluePrints On-line*, Sun Microsyst., Santa Clara, CA, USA, 2004. [Online]. Available: http://home.mit.bme.hu/~meszaros/edu/oprendszerek/segedlet/unix/x_memoriakezeles/solaris_Multiple_Page_Size_Support.pdf
[14] Y. A. Khalidi, G. R. Anderson, S. A. Chessin, S. I. Kong, C. E. Narad, and M. Talluri, "Virtual address to physical address translation cache that supports multiple page sizes," U.S. Patent 5 479 627, Dec. 26, 1995.
[15] S. Winwood, Y. Shuf, and H. Franke, "Multiple page size support in the Linux kernel," in *Proc. Ottawa Linux Symp.*, 2002, pp. 573–593.
[16] R. Ausavarungnirun, J. Landgraf, V. Miller, S. Ghose, J. Gandhi, C. J. Rossbach, and O. Mutlu, "Mosaic: Enabling application-transparent support for multiple page sizes in throughput processors," *ACM SIGOPS Oper. Syst. Rev.*, vol. 52, no. 1, pp. 27–44, Aug. 2018.
[17] F. Guvenilir and Y. N. Patt, "Tailored page sizes," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 900–912.
[18] E. Lee, S. Hoon Yoo, and H. Bahn, "Design and implementation of a journaling file system for phase-change memory," *IEEE Trans. Comput.*, vol. 64, no. 5, pp. 1349–1360, May 2015.
[19] E. Lee, H. Bahn, S. Yoo, and S. H. Noh, "Empirical study of NVM storage: An operating system's perspective and implications," in *Proc. IEEE 22nd Int. Symp. Modeling, Anal. Simul. Comput. Telecommun. Syst.*, Sep. 2014, pp. 405–410.
[20] *Memory Access Traces*. Accessed: Feb. 28, 2021. [Online]. Available: https://github.com/oslab-ewha/memtrace
[21] D. Shin, K. Cho, and H. Bahn, "File type and access pattern aware buffer cache management for rendering systems," *Electronics*, vol. 9, no. 1, p. 164, Jan. 2020.
[22] 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.*, vol. 63, no. 9, pp. 2187–2200, Sep. 2014.
[23] *Protected and Persistent RAM Filesystem*. Accessed: Feb. 28, 2021. [Online]. Available: http://pramfs.sourceforge.net
[24] N. K. Edel, D. Tuteja, E. L. Miller, and S. A. Brandt, "MRAMFS: A compressing file system for non-volatile RAM," in *Proc. 12th IEEE Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst.*, Oct. 2004, pp. 596–603.
[25] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better I/O through byte-addressable, persistent memory," in *Proc. ACM Symp. Oper. Syst. Princ. (SOSP)*, Oct. 2009, pp. 133–146.
[26] E. Lee, J. E. Jang, T. Kim, and H. Bahn, "On-demand snapshot: An efficient versioning file system for phase-change memory," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2841–2853, Dec. 2013.
[27] X. Wu and A. L. N. Reddy, "SCMFS: A file system for storage class memory," in *Proc. Int. Conf. Supercomput. (SC)*, 2011, pp. 1–11.
[28] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson, "NV-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories," in *Proc. 16th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2011, pp. 105–118.
[29] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: Lightweight persistent memory," in *Proc. 16th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2011, pp. 91–104.
[30] I. Peng, M. McFadden, E. Green, K. Iwabuchi, K. Wu, D. Li, R. Pearce, and M. Gokhale, "UMap: Enabling application-driven optimizations for page management," in *Proc. IEEE/ACM Workshop Memory Centric High Perform. Comput. (MCHPC)*, Nov. 2019, pp. 71–78.
[31] J. Yang, D. B. Minturn, and F. Hady, "When poll is better than interrupt," in *Proc. Conf. File Storage Technol. (FAST)*, 2012, p. 3.
[32] A. M. Caulfield, A. De, J. Coburn, T. I. Mollov, R. K. Gupta, and S. Swanson, "Moneta: A high-performance storage array architecture for next-generation, non-volatile memories," in *Proc. IEEE/ACM Symp. Microarchitecture (Micro)*, Dec. 2010, pp. 385–395.

YUNJOO PARK received the B.S. degree in computer science and engineering from Ewha Womans University, South Korea, in 2015, where she is currently pursuing the Ph.D. degree in computer science and engineering. Her research interests include operating systems, storage systems, embedded systems, software platform technologies, cloud computing, and emerging storage systems.

HYOKYUNG BAHN (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Seoul National University, in 1997, 1999, and 2002, respectively. He is currently a Full Professor of computer science and engineering with Ewha Womans University, Seoul, Republic of Korea. He has published more than 100 papers in leading conferences and journals including USENIX FAST, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and ACM *Transactions on Storage*. His research interests include operating systems, caching algorithms, storage systems, embedded systems, system optimizations, and real-time systems. He received the Best Paper Awards at the USENIX Conference on File and Storage Technologies in 2013.