SysMon: Monitoring Memory Behaviors via OS Approach

Mengyao Xie[1,2,3], Lei Liu[1,2 *], Hao Yang[1,2,3], Chenggang Wu[2] and Hongna Geng[1,2,3]

[1] Sys-Inventor Research Group, [2] State Key Lab. Of Computer Architecture, ICT, CAS
[3] University of Chinese Academy of Sciences
Email:{xiemengyao,liulei2010}@ict.ac.cn

Abstract. To capture and analyze applications' memory behaviors with low overhead plays a vital role in managing and scheduling memory resources on modern computer systems. In this paper, we re-design SysMon based on [13, 14], which is an OS-level memory behaviors monitoring module in existing OS, and modify its several core components to meet the challenges of higher efficiency and accuracy. SysMon can be used without offline profiling, instrumentation or configuring complex parameters. We evaluate SysMon by making a great deal of experiments on SPECCPU 2006 [7], Memcached [1] and Redis [6]. The experimental results show that, by using SysMon, we can efficiently capture the memory footprint, write/read operations, hot/cold features, re-use time, bank hotness/bank balance, etc. Besides, we collect the memory access behaviors in the configuration of different sampling intervals, and draw a conclusion that using a 3 seconds interval can obtain information accurately with low overhead. Finally, to reduce the scanning overhead during samplings, SysMon adopts a randomization method, and scans only a portion of pages. Experiments show that the sampling overhead can be reduced by 44.42% on average while guaranteeing the accuracy of sampling.

Keywords: memory behaviors, system monitor tool, random sampling, sampling interval.

1 INTRODUCTION

Allocating, managing and scheduling of memory resources have always been a major and very challenging subject on modern computer systems. With the emerging of big data and cloud computing, fast-growing memory footprint and energy consumption, high demand for Quality of Service (QoS) and throughput, etc. have brought new challenges to memory management [20-23, 25]. Especially, it may result in the severer memory access conflict with high probability when multiple applications are running in parallel. Many previous studies [8, 9, 13, 16, 19, 26, 27] show that it is important for operating systems to efficiently manage data with low overhead. In order to achieve this goal, there are many factors need to be considered to manage memory system efficiently, such as the different characteristics of data (e.g., write/read operations, hot/cold features), memory access hotness, re-use time, etc. Thus, an effective memory management policy is expected to accurately detect the applications' memory behaviors and schedule memory resources accordingly.

The existing program analysis tools like Intel's dynamic binary instrumentation framework Pin [5] can be used to create Pintools to perform program analysis on user space applications on Linux, Windows and OS X*. However, instrumentation con-

This work is supported by NSF of China under grants No. 61502452 (PI: Lei Liu).
* Lei Liu is the corresponding author.

sumes system resources, and thus increases the profiling overhead when analyzing the applications' behaviors. Another tool, Oprofile [2], is a performance counter monitor tool that monitors the running applications based on Performance Monitoring Unit (PMU). However, Oprofile and other performance counter monitor tools like PAPI [3] and perfmon2 [4] require underlying hardware support (i.e., PMU). And many of them cannot fully support the newer architectures because of the diversification of the hardware architecture.

Compared with above approaches, SysMon [13, 14] is an efficient and lightweight application access behaviors monitor tool, which is a module that integrated into the kernel. It can be used on any version of Linux kernel without instrumentation, configuring complex parameters, or extra underlying hardware support. SysMon has good compatibility, stability, and scalability. However, in practice, some studies further show that the overhead brought by SysMon is heavy for some applications with much higher memory footprint and the sampling interval is hard to be determined to balance the overhead and accuracy in many real cases. To address these concerns, we redesign SysMon and make the following contributions in this paper:

- We optimize SysMon's sample method by adopting random sampling rather than traversing the page table to sample each page. The experimental results show that the sampling overhead can reduce 44.42% on average while ensuring the sample effect.
- We collect the memory access information under the configuration of different sampling intervals. By analyzing the information, we draw a conclusion that using a 3 seconds interval can obtain information accurately with low overhead.
- By using SysMon, we study a large number of workloads, and analyze their characteristics, including SPECCPU2006 [7], Memcached [1] and Redis [6].

We open sourced SysMon. The full code of SysMon is available at:
https://github.com/Sys-Inventor-Research-Group-ICT/Sysmon

2 BACKGROUND

2.1 *__access_bit* and *__dirty_bit*

Starting from Linux v2.6.11, 64-bit Operating System (OS) adopts the organizational form of the four-layer page table, which is represented in Figure 1. Each item in Page Global Directory (PGD) points to a Page Upper Directory (PUD), and each entry in PUD points to a Page Middle Directory (PMD), and then, each item in PMD points to a PTE.

The *__access_bit* in page table entry (PTE) can be used to indicate whether the page is accessed [11, 18]. 0 represents the page has not been accessed; while 1 means accessed (we define these pages as hot pages in this paper). And for the *__dirty_bit*, it can represent whether the page is modified. Similar to the *__access_bit*, when the *__dirty_bit* is equal to 0, it means there is no write operation happened to that page.

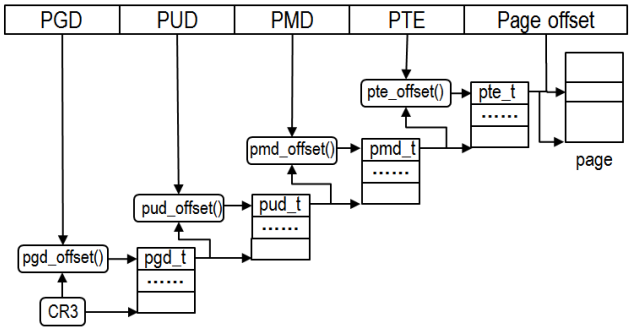

Fig. 1. Four-layer page table under 64-bit operating system.

2.2 Address mapping

Prior research [28] shows that mainstream computer systems' address mapping can be detected by the software method. For example, as shown in Figure 2, bank bits are divided into two parts. Part I is independent, and part II is overlapped with cache bits. Figure 2 (a) presents Intel i7-860 processor that equips with a 16-way set associative 8MB last level cache (LLC) and 8GB DDR3 main memory system, and it's bank bits are 13-15, 21 and 22 bits; In Figure 2 (b), Intel Xeon 5600 processor, with 16-way set associative 12MB LLC and 32GB DDR3 main memory, whose bank bits are 13, 14, 20 and 21 bits. For the configuration (a), 5 bank bits can index $2^5 = 32$ banks ranging from bank 0 to bank 31.

3 DESIGN AND IMPLEMENTATION

3.1 Overview

SysMon captures application behaviors dynamically such as memory footprint, page access frequency, re-use time of pages, memory utilization, etc. The information is collected online without offline profiling and does not need hardware performance counters.

The design of SysMon is based on the three following principles:

Principle 1: Compatibility. SysMon is integrated in the Linux kernel as a kernel module to monitor page-level application activities. It is reliable, portable and suitable

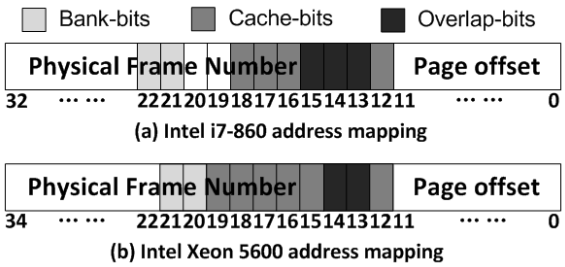

Fig. 2. Address mapping.

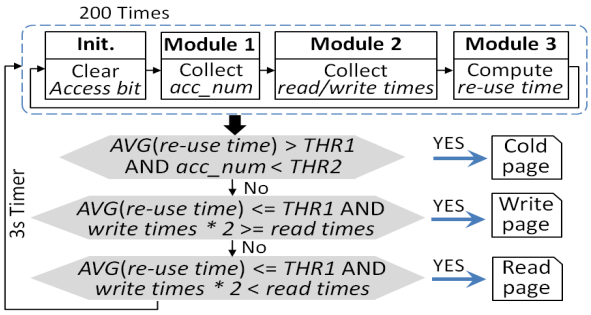

Fig. 3. SysMon-based online page classification algorithm.

for any version of Linux kernel.

Principle 2: Low overhead. SysMon is a lightweight online tool that monitors applications in the real time. The overhead is mainly caused by scanning application's page table. Through the random scanning optimization method, which is introduced in detail at Chapter 5, SysMon greatly reduces the scanning overhead by 44.42% on average.

Principle 3: Efficiency. It is important for a monitoring tool that does not slow the responses to the applications' access requests. Our experiments show that 100μs is enough to collect sufficient information while incurring a negligible delay.

Except for monitoring the single application, SysMon can also monitor multiple applications that are executed in parallel. By analyzing the information captured by SysMon, we can make an accurate prediction of a running workload's memory characteristics, and use an appropriate memory management policy.

As shown in Figure 3, we take a page classification algorithm as an example to introduce the modules of SysMon. The information in the dashed box is collected by SysMon, and *acc_num* records the page's total number of accesses in a given period, *read/write times* are being used to indicate the number of read/write operations on the pages during samplings. *Re-use time* is a variable to represent the page's temporal locality. Based on the information in the dashed box in Figure 3, we classify the pages into three categories: write page, read page and cold page. In our experiments, *THR1* is 20 and *THR2* is 10. The detailed information about pages' characteristics can guide the data placement and data movement among the DRAM Banks to improve the overall performance.

In the next section, we will introduce the modules of SysMon one by one.

3.2 Module 1: Collecting page access frequency

In the current version, the time interval between two sampling periods is 3 seconds in our system. To reduce the error efficiently, 200 samplings are executed in one sampling period (i.e., 3s), but note that the time cost of 200 samplings is far less than 3s (100 ns in most cases). Each sampling contains two loops. Firstly, SysMon clears pages' *__access_bit* by the *pte_mkold()* kernel function; and secondly, SysMon checks the pages' *__access_bit* in the second loop. If the *__access_bit* is still 0, it means the page has not been accessed in this sampling; otherwise, this page has been

Algorithm_1: Calculate Access Frequency and Write / Read Times of Pages
Output: (1) hot_page [ ]; (2) write_times [ ]; (3) read_times [ ]

```
1. FOR i FROM 0 TO ITERATIONS DO // default 200
2.     Clear __access_bit, __dirty_bit of all pages;
3.     FOR each page P IN an application DO
4.         Obtain PTE of page P via P's address;
5.         IF pte_present (PTE) THEN      // P is in the memory
6.             IF pte_young (PTE) THEN    // P has been accessed
7.                 hot_page [P] ← hot_page [P]; // access frequency
8.                 IF ( pte_dirty (PTE) ) THEN    // write operation
9.                     write_times [P] ← write_times [P] + 1;
10.                ELSE   // read operation
11.                    read_times [P] ← read_times [P] + 1;
12.                END IF
13.            END IF
14.        END IF
15.    END FOR
16. END FOR
17. RETURN hot_page [ ], write_times [ ], read_times [ ];
```

* hot_page [ ] is an array representing access frequency of each page.
* write_times [ ] is an array that records write times of the pages.
* read_times [ ] is an array that records read times of the pages.

accessed during this sampling.

To locate the PTE and check the *__access_bit* of each page during the samplings, SysMon needs to lookup virtual address layer by layer (see Figure 1). In consideration of the fact that all pages targeted by a request are virtually contiguous, most of their PTEs are adjacent. It means that SysMon only needs to obtain the first page's PTE from the page table root; for each of the remaining pages, we can get their PTEs by adding a fixed offset without starting from PGD [12]. Traversing like this can reduce the sampling overhead.

For the running applications, Algorithm_1 shows the pseudo-code for obtaining the page access frequency. In the first loop, SysMon clears all pages' *__access_bit* (Line 2); and then, check the *__access_bit* using function *pte_young()* (Line 6).

After 200 samplings, SysMon will calculate the total number of accesses of each page, and grade pages according to the page "heat" (i.e., the number of accesses). Classification standard in our experiments is shown in Table 1. It can be adjusted according to the characteristics of workloads. In addition, SysMon can calculate the memory footprint of the running workload.

3.3 Module 2: Write/read operations statistics

SysMon dynamically monitors the write/read operations of hot pages during samp-

Table 1. Classification standard for page "heat".

The number of accesses	Page "Heat"	The number of accesses	Page "Heat"
Larger than 200	Very High	64 ~ 100	Low
150 ~ 200	High	10 ~ 64	Lower
100 ~ 150	Medium	Less than 10	Very low

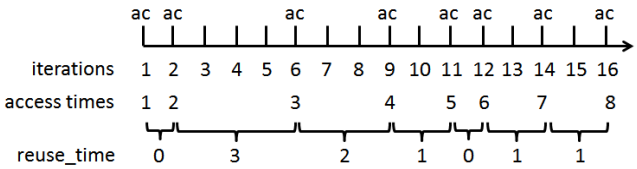

Fig. 4. Re-use time of one page.

lings. In the page classification process (see Figure 3), we give write operations a heavier weight as write operations are more expensive than read operations in memory system (i.e., empirical value is 2 since write operations need to read data, modify and write back to the memory, causing a longer latency than read operations [29]). And this value can be adjusted according to the specific environments and configurations.

Algorithm_1 shows how to calculate the write/read times of each page. SysMon clear the *__access_bit* and *__dirty_bit* in the first loop (Line 2); and in the second loop, if *pte_dirty()* returns 1, it means write operation occurs. Otherwise, a read operation is detected (Line 8-Line 12). Moreover, SysMon can also record that, compared with the last sampling, the number of write pages converting into read pages and the number of read pages converting into write pages. It is meaningful for the data placement that distinguishes the page is a write domain page or a read domain page.

3.4 Module 3: Re-use time statistics

In order to calculate re-use time of a page, SysMon monitors whether this page is accessed in each sampling, and uses an array to record the interval between the two accesses, this is so-called "re-use time" of that page. Figure 4 denotes the re-use time of the selected page, where *iterations* means the samplings, and *access times* records the picked page's access times. Algorithm_2 describes how to calculate the re-use time of a page. SysMon checks the *__access_bit*, if the page is accessed, the number of accesses *times* adds 1; if not, the re-use "distance" between last access and next access increases 1 (Line 6-Line 10).

Algorithm_2: Calculate Re-Use Time
Input: The NO. of page P **Output:** reuse_time [ ]

```
1. FOR i FROM 0 TO ITERATIONS // default 200
2. DO
3.   Clear __access_bit of page P;
4.   Obtain PTE of page P via P's address;
5.   IF pte_present (PTE) THEN   // P is in the memory
6.     IF pte_yong (PTE) THEN   // P has been accessed
7.       times ← times + 1;
8.     ELSE   // P has not been accessed
9.       reuse_time [times] ← reuse_time [times] + 1;
10.    END IF
11.  END IF
12. END FOR
13. RETURN reuse_time [ ];
```

* times is used to record the access times of the page P.
* reuse_time [ ] records the re-use time of page P.

Algorithm_3: Bank Hotness Statistics
Output: bank_hotness [ ]

```
1. #define BANKBITS_1   (0x600)   // 21, 22 bits
2. #define BANKBITS_2   (0xE)   // 13, 14, 15 bits
3. #define PAGE_TO_BANK   ((page_to_pfn (page) & (BANKBITS_1))
            >> 6 | ((page_to_pfn (page) & (BANKBITS_2)) >> 1
4. FOR i FROM 0 TO ITERATIONS DO// default 200
5.    Clear __access_bit of all pages;
6.    FOR each page P IN an application DO
7.       Obtain PTE of page P via P's address;
8.       IF pte_present (PTE) THEN   // P is in the memory
9.          IF pte_yong (PTE) THEN   // P has been accessed
10.            bank_id ← PAGE_TO_BANK (pte_page (PTE));
11.            bank_hotness [bank_id] ← bank_hotness [bank_id] + 1;
12.         END IF
13.      END IF
14.   END FOR
15. END FOR
16. RETURN bank_hotness [ ]
```

* PAGE_TO_BANK is a Macro definition that returns the bank_id according to Page Frame Number (PFN).
* bank_hotness [ ] records the number of hot pages in each bank.

The pages to be monitored are chosen randomly before samplings. By doing so, SysMon guarantees that there is less deviation when collecting re-use time information during samplings. Page-level re-use time information is an important factor that reflects the application access behaviors, which represents the temporal locality of the pages. By analyzing the re-use time, we can quantify how quickly the particular pages will be accessed again. Taking re-use time into account can accurately reflect the page access trend and the applications' overall memory access trend during the period of time.

3.5 Module 4: Bank hotness statistics

The main memory system is composed of several DRAM banks that are shared by multiple running processes. When several requests from different process falling on the same DRAM bank, the access conflict occurs, and these requests have to be handled in a sequential order. This causes row buffer thrashing and a longer access delay, and declines the overall performance of the system. Therefore, it is the foundation of further optimizing memory scheduling algorithms to clearly understand the bank hotness/balance information among several DRAM banks.

As illustrated in Algorithm_3, SysMon calculates the number of hot pages in each bank. *PAGE_TO_BANK* is a macro definition that can extract the bank bits and obtain the bank id (Line 3). Note that Algorithm_3 is implemented with channel interleaving under the configuration of Figure 2 (a). When the entire bandwidth demand is larger than 2GB/s, channel partition is more effective and can avoid significant performance degradation [15]. In the case above, since there are 64 banks in the memory system (32 banks/per channel), *PAGE_TO_BANK* should simultaneously extract channel bit and bank bits to calculate the bank id.

4 OPTIMIZATION

For the applications that need large memory footprint, to reduce the scanning overhead during samplings, SysMon randomly scans a portion of pages instead of traversing all the Virtual Memory Areas (VMAs). As illustrated in Figure 5, SysMon scans 5% pages in our experiments. Before sampling, SysMon generates a random number as the sampling's starting point within a VMA by using function *get_random_bytes()*. The sampling interval of pages can be calculated by scanning ratio (i.e., 1 / 0.05 = 20 in our experiments). The scanning ratio can be adjusted as required.

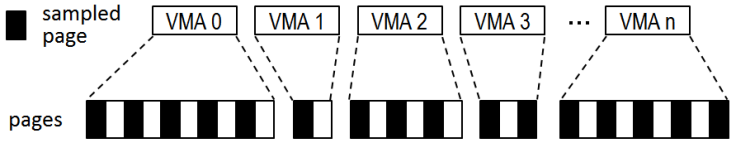

Fig. 5. SysMon samples a portion of pages to analyze the applications' behaviors. Note that the sampling fraction here is only for illustration purpose. In our experiments, we sample 5% of pages during each sampling.

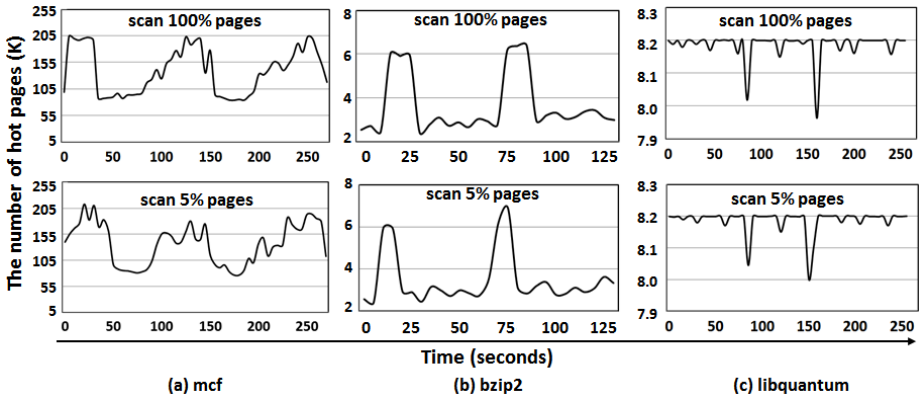

Fig. 6. The number of hot pages by using random sampling method.

To reduce the error efficiently, SysMon uses different random numbers before each sampling. After 200 samplings, all the pages can be covered. We adopt equal interval sampling (i.e., sample page 0, 20, 40, 60…) instead of completely random design (i.e., generate random numbers constantly as the page number during samplings). It is because if we use the second method, we have to record all the random numbers, so the space overhead will increase linearly as the memory footprint increases; it is contrary to the intention of "randomization to reduce the sampling overhead", and not worth the candle.

Our experiments show that sample 5% pages can accurately reflect the applications' memory access trend, the ratio of hot pages, etc. Figure 6 gives several examples of benchmarks. Experiments show that randomization can reduce the scanning overhead by 33.12% at least (tonto), 47.89% at most (Memcached), and 44.42% on average.

5 EVALUATION

5.1 How to run SysMon

We study SysMon on the configuration of Figure 2 (a). To run SysMon, we firstly need to write a *Makefile* file. Each source file (i.e., *.c) corresponds to a line "obj-m += *.o" in the *Makefile*. After using *make* command to compile the source files, we then use *insmod* **.ko* command to insert the module into the kernel. Finally, use *dmesg* to output the results.

5.2 Benchmarks

We evaluate SysMon with diverse workloads, including SPECCPU2006, widely used Memcached with data from Twitter and Redis. SPECCPU2006 benchmark is an industry-standardized, CPU-intensive benchmark suite. The widely used Memcached is a distributed memory object caching system. It is an in-memory key-value store for small chunks of arbitrary data from results of database calls, API calls, or page rendering. Redis is a popular NoSQL database and is single-threaded. Redis has no file I/O after loading the dataset into memory.

5.3 Experimental results

Memory footprint and write/read operations. Figure 7 shows the benchmarks' average normalized portion of different types of pages (i.e., write page, read page and cold page). It can be seen from Figure 7 that more than 80% pages of omnetpp, sjeng, lbm and GemsFDTD are hot pages; more than 90% pages of lbm are write pages. For bzip2 and namd, less than 10% pages are hot pages. As for Memcached and Redis, though their memory footprints are large, the portion of hot pages/active pages is not that large.

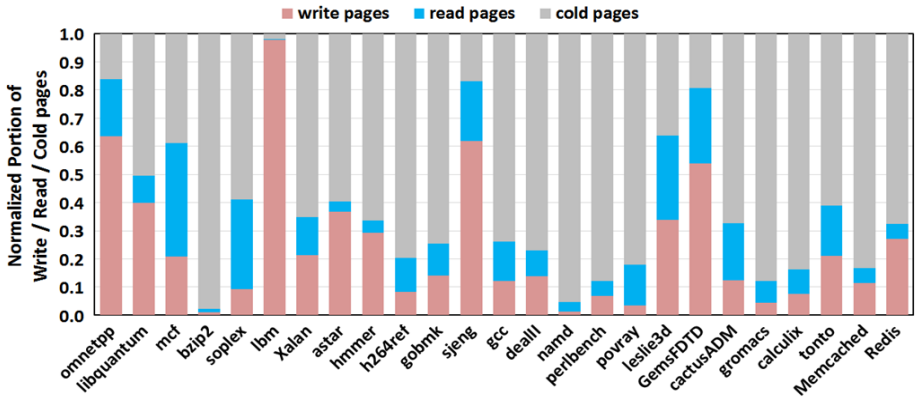

Fig. 7. Normalized portion of the three types of pages of different benchmarks.

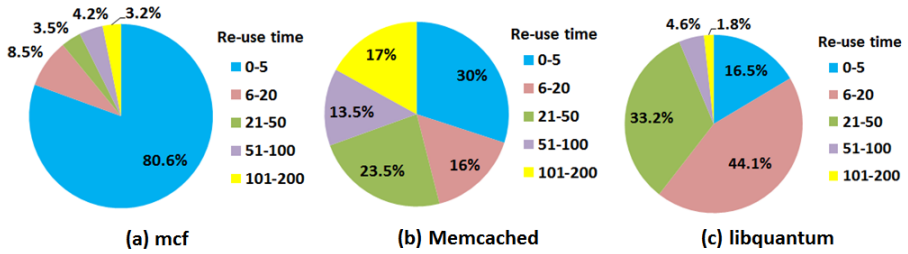

Fig. 8. Normalized portion of different re-use time sections.

Re-use time. We tested all the benchmarks and observed that there are two categories can be classified by the re-use time characteristics. One is that most re-use times are relatively small; the other is the re-use times are evenly distributed in different sections.

Figure 8 represents the portion of different re-use time sections. Figure 8 (a) shows that for mcf, 80.6% re-use time (i.e., re-use distance) is less than 5, and only 7.4% re-use time is larger than 50 within 200 samplings; it means that the memory access for mcf is very intensive. Libquantum (Figure 8 (c)) is similar to mcf, most re-use times are between 0 and 20, only 6.4% re-use time is larger than 50. As for Memcached (Figure 8 (b)), the re-use time distribution is more balanced, which indicated that memory access is not that intensive compared with mcf and libquantum.

Bank hotness. Figure 9 illustrates the normalized hot page number (i.e., bank hotness) within each DRAM bank. By exploring the bank hotness of all benchmarks, we found that the hot page distribution is not balanced in many cases. Taking Memcached as an example, the hottest bank (bank 31) has 531 more hot pages than the coldest bank (bank 15). Besides, we randomly choose two workloads and test their bank hotness. To eliminate the bank unbalance, L. Liu et al. [17] proposes a page-coloring based bank-level partition mechanism, which allocates specific DRAM banks to specific threads.

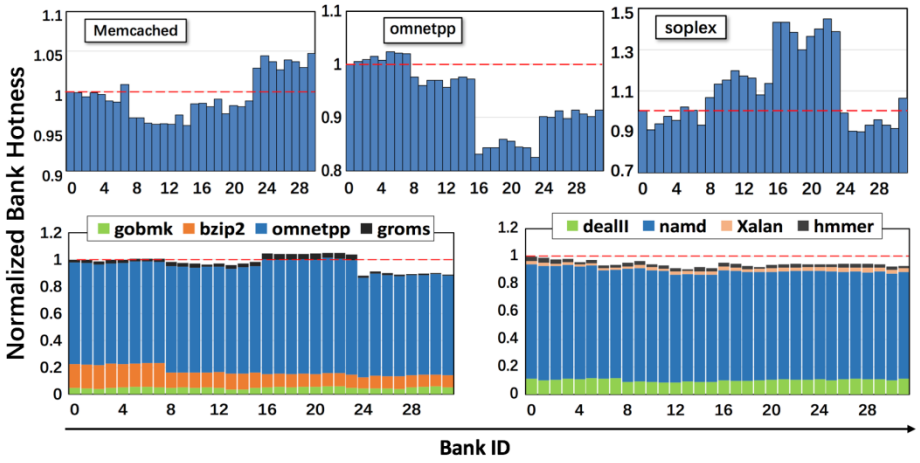

Fig. 9. Normalized bank hotness of single benchmark and multi-benchmarks.

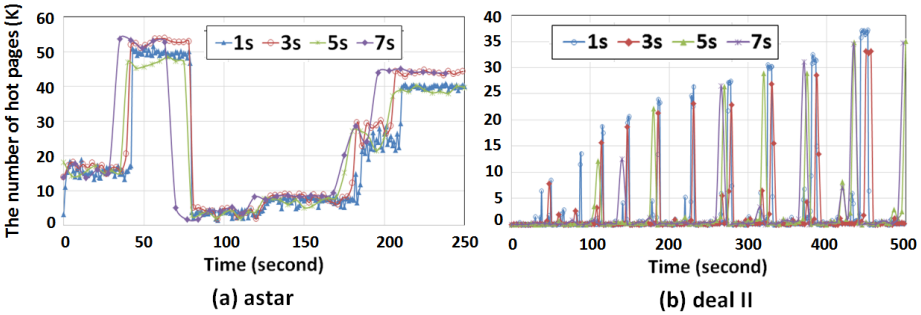

Fig. 10. The number of hot pages under the configuration of different sampling intervals.

5.4 Sampling interval

In our experiments, the sampling interval between two sampling periods is set to 3 seconds. In terms of the time interval, we are challenged by a question: how much the interval should we use to obtain the applications' memory access information with low overhead and good accuracy? To study the relation between sampling accuracy and sampling interval, we test the hot page numbers of all benchmarks by using different intervals (i.e., 1s, 3s, 5s, and 7s). Due to the space limitation, we show two benchmarks in Figure 10. It can be seen that the variation trends of hot page numbers are similar no matter how much the time interval is. Note that the smaller interval, the higher overhead, so we choose 3 seconds in our platform to balance the accuracy and overhead. By doing so, we can guarantee the accuracy while not costing so much overhead.

6 RELATED WORK

Many previous researches [10, 24] performed profiling in the real time by the support of hardware performance counters. In this paper, without hardware supports, SysMon obtains memory access behaviors online via OS approach, and is able to collect the page-level re-use time, bank balance/hotness, and the write/read characteristics [14, 16]. The captured information is critical for the memory management on hybrid DRAM-NVM system [13, 19, 22].

7 CONCLUTION

This paper re-designs SysMon as a Linux kernel module to meet the challenges on monitoring large memory footprint applications. To balance the sampling overhead and accuracy, we adopt a random sampling method and explore the appropriate sampling interval. Experiments show that 44.42% sampling overhead on average can be reduced by using random sampling method. We capture a large number of benchmarks' memory behaviors including page access frequency, write/read and hot/cold features, re-use time and bank balance/hotness by using SysMon.

References

[1] Memcached. http:// memcached.org
[2] Oprofile. http://oprofile.sourceforge.net/news/
[3] PAPI. http://icl.utk.edu/papi/
[4] Perfmon2. http://perfmon2.sourceforge.net/
[5] Pin.https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool
[6] Redis. http://redis.io/
[7] SPECCPU2006. http://www.spec.org/cpu2006
[8] Christina Delimitrou, Christos Kozyrakis. “Quasar: Resource-Efficient and QoS-Aware Cluster Management.” In ASPLOS, 2014.
[9] N. Duong, D. Zhao, T. Kim, et al. “Improving Cache Management Policies Using Dynamic Reuse Distances.” In MICRO, 2012.
[10] A. Jaleel, H. H. Najaf-Abadi, S. Subramaniam, S.C. Steely, and J. Emer, “CRUISE: Cache Replacement and Utility-Aware Scheduling,” In ASPLOS, 2012.
[11] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and Emmett Witchel. "Coordinated and efficient huge page management with ingens." In OSDI, 2016.
[12] F. X. Lin, X. Liu. “Memif: Towards programming heterogeneous memory asynchronously.” In ASPLOS, 2016.
[13] L. Liu, H. Yang, Y. Li, M. Xie, L. Li. C. Wu. “Memos: A Full Hierarchy Hybrid Memory Management Framework.” In ICCD, 2016.
[14] L. Liu, Y. Li, C. Ding, H. Yang, C. Wu. “Rethinking Memory Management in Modern Operating System: Horizontal, Vertical or Random?” In TC, 2016.
[15] L. Liu, Z. Cui, Y. Li, Y. et al. "BPM/BPM+: Software-based Dynamic memory partitioning mechanisms for mitigating DRAM Bank-/Channel-level interferences in multicore systems." In TACO, 2014.
[16] L. Liu, Y. Li, Z. Cui, C. Wu, et al. “Going Vertical in Memory Management: Handling Multiplicity by Multi-policy.” In ISCA, 2014.
[17] L. Liu, Z. Cui, M. Xing, C. Wu, et al. "A software memory partition approach for eliminating bank-level interference in multicore systems." In PACT, 2012.
[18] S. Lee, Hyokyung Bahn, and Sam H. Noh. "Clock-dwf: A write-history-aware page replacement algorithm for hybrid pcm and dram memory architectures." In TC, 2014.
[19] L. Liu, M. Xie and H. Yang. “Memos: Revisiting Hybrid Memory Management in Modern Operating System.” In arXiv:1703.07725, 2017.
[20] F. Lv, L. Liu, et al. “WiseThrottling: A New Asynchronous Task Scheduler for Mitigating I/O Bottleneck in Large-Scale Datacenter Servers.” In J. of Supercomputing, 2015.
[21] F. Lv, H. Cui, L. Wang, L. Liu, et al. “Dynamic I/O-Aware Scheduling for Batch-Mode Applications on Chip Multiprocessor Systems of Cluster Platforms.” In JCST, 2014.
[22] L. Liu. “Tackling Diversity and Heterogeneity by Vertical Memory Management.” In arXiv:1704.01198, 2017.
[23] Y. Liang and X. Li. “Efficient Kernel Management on GPUs.” In TECS, 2017.
[24] H. T. Mai, K. H. Park, H. S. Lee, C. S. Kim, M. Lee, S. J. Hur, “Dynamic Data Migration in Hybrid Main Memories for In-Memory Big Data Storage,” In ETRI Journal, 2014.
[25] O. Mutlu. Main memory scaling: Challenges and solution directions[M]//More than Moore Technologies for Next Generation Computer Design. Springer New York, 2015: 127-153.
[26] Rixner S, Dally W J, Kapasi U J, et al. Memory access scheduling[C]//ACM SIGARCH Computer Architecture News. ACM, 2000, 28(2): 128-138.
[27] G. Sun, et al. “Statistical Cache Bypassing for Non-Volatile Memory.” In TC, 2016.
[28] Mi. W, Feng. X, Xue. J, et al. “Software-hardware cooperative DRAM bank partitioning for chip multiprocessors.” In NPC, 2010.
[29] Kim. Y, Seshadri. V, Lee. D, Liu. J, and Mutlu. O. “A case for exploiting subarray-level parallelism (SALP) in DRAM”. ACM SIGARCH Computer Architecture News, 40(3), 368-379, 2012.