M.S. THESIS

Optimizing Memory Swap Overhead Through Pattern Analysis

메모리 스왑 패턴 분석을 통한 스왑 시스템 최적화

Hyerin Chung

FEBRUARY 2021

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

M.S. THESIS

Optimizing Memory Swap Overhead Through Pattern Analysis

메모리 스왑 패턴 분석을 통한 스왑 시스템 최적화

Hyerin Chung

FEBRUARY 2021

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Optimizing Memory Swap Overhead Through Pattern Analysis

메모리 스왑 패턴 분석을 통한 스왑 시스템 최적화

지도교수 염 헌 영

이 논문을 공학석사 학위논문으로 제출함

2020 년 12 월

서울대학교 대학원

컴퓨터 공학부

정 혜 린

Hyerin Chung의 공학석사 학위논문을 인준함

2021 년 01 월

위 원 장 장병탁 (인)

부위원장 염헌영 (인)

위 원 엄현상 (인)

Optimizing Memory Swap Overhead Through Pattern Analysis

메모리 스왑 패턴 분석을 통한 스왑 시스템 최적화

지도교수 염 헌 영

이 논문을 공학석사 학위논문으로 제출함

2020 년 12 월

서울대학교 대학원

컴퓨터 공학부

정 혜 린

Hyerin Chung의 공학석사 학위논문을 인준함

2021 년 01 월

위 원 장	장병탁	(인)
부위원장	염헌영	(인)
위 원	엄현상	(인)

Abstract

The use of memory is one of the key parts of modern computer architecture (Von Neumann architecture) but when considering limited memory, it could be the most lethal part at the same time. Advances in hardware and software are making rapid strides in areas such as Big Data, HPC and machine learning and facing new turning points, while the use of memory increases along with those advances. In the server environment, various programs share resources which leads to a shortage of resources. Memory is one of those resources and needs to be managed. When the system is out of memory, the operating system evicts some of the pages out to storage and then loads the requested pages in memory. Given that the storage performance is slower than the memory, swap-induced delay is one of the critical issues in the overall performance degradation. Therefore, we designed and implemented a "swpTracer" to provide visualization to trace the swap in/out movement. To check the generality of the tool, we used mlock to optimize 429.mcf of Spec CPU 2006 based on the hint from swpTracer. The optimized program executes 2 to 3 times faster than the original program in a memory scarce environment. The scope of the performance improvement with previous system calls decreases when the memory limit increases. To sustain the improvement, we build a swap- prefetch to read ahead the swapped-out pages. The optimized application with swpTracer and swap-prefetch consistently exceeds the performance of the original code by 1.5x.

Keywords: Memory, Swap, Visualization, Profile, Optimization

Student Number: 2019-25099

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern workloads are memory intensive. Machine learning, High Performance Computing (HPC) workloads, Bioinformatics, etc. have shown enhanced results with the support of advanced technology. However, these applications require a huge capacity and high bandwidth of memory.

Memory is one of the key components that loads everything to operate the system. As the amount of data to be processed increases, more memory is needed which eventually becomes a bottle neck. Compared to the other components of the computer, the scale of improvement in main memory is relatively small. Besides DRAM, new techniques and products such as NVM and Persistent Memory were suggested to complement each memory hierarchy. Still, there is high latency between the main memory and storage.

Memory management is an important job for an operating system, or else, the whole system will be blocked. Swap is one of the tricks for an operating system to protect the priceless resource even though it contains latency during the page transfer. The latency between the memory and storage is more critical

than the latency between the CPU and memory, but this also means that the burden has just been doubled.

Because systems seem to suffer from a depletion of resources less nowadays compared to the past few decades, swapping in a system might only be an issue at some point. Nevertheless, in the following scenarios, the use of swap is obviously necessary.

1) When sufficient memory is not available because of a temporal memory-intensive workload, it is difficult to evict a file-backed page.

2) Hibernation for energy-limited environments.

3) System environments where responsiveness is critical.

4) Embedded environments with limited resources.

A good rule of thumb for swap in memory is that it should be differently configured based on the scenarios and system resources. To solve the memory shortage, the operating system exchanges pages between the main memory and storage. If a page fault occurs because there is no page to access the page table, the page fault handler reads the page from the hard disk and stores it in memory. Because hard disks have a very different performance compared to the main memory, the overall run time is increased by programs waiting to perform a memory swap.

Understanding the memory usage pattern of a target application and selecting an adequate memory management policy are one way to enhance the performance. When complicated programs are split and analyzed, there are common memory access patterns among applications. For example, a matrix vector multiplication is the basis of big data analysis and machine learning. Meanwhile, graph computations are one of the typical irregular access pattern workloads which are used widely from network systems to web based applica-

tions. Considering the use of large amounts of data in recent computational workloads and the regularity in memory access patterns, analyzing programs can improve performance.

In this study, we designed and implemented swpTracer, a tool that visualizes memory swap generated during program performance in an offline manner. It provides a hint to optimize the swap operation of the workloads. The purpose of swpTracer is to provide an insight to the swap so the transfers between the memory and storage can decrease when there is insufficient memory. Especially, it focuses on helping programmers to trace down the features that are involved with the swap during the process. To obtain the basic information about each object, swpTracer parses the entire code statically and inserts a hint instrument into the code before carrying out the program. swpTracer focuses on the workloads with data locality. By analyzing the flow of the swap operation, users can determine the overview of the program's flow from the memory to the swap area. swpTracer analyzes using the kernel log and memory map information to extract the object to be optimized and visualizes the results of the analysis. swpTracer itself may not be a solution for the problem, but it will provide a hint to the programmers to help solve the problem. The memory access pattern will be key to optimizing performance in program deployment. The contributions of this paper are as follows:

- swpTracer visualizes the swap out memory latency.
- By automatically adding a few lines online in the compiling stage, it could provide information about fault-able pages.
- swpTracer not only provides the native information of a faulted address but also provides simple statistical information about the memory latency.

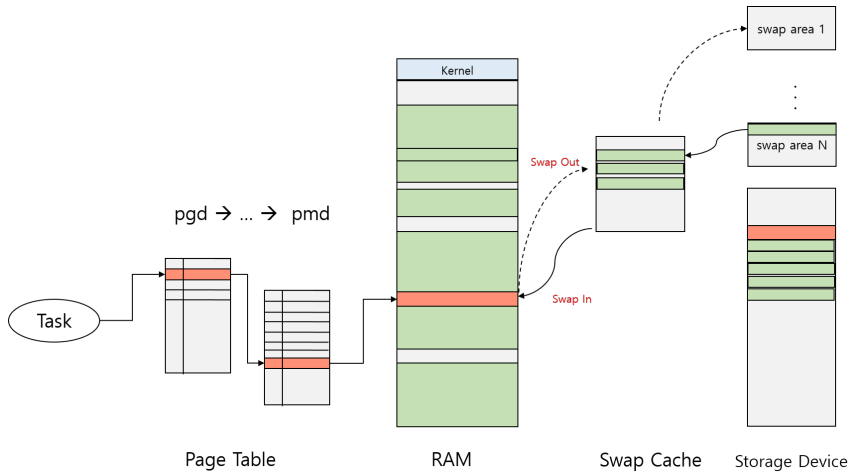
Chapter 2

Background

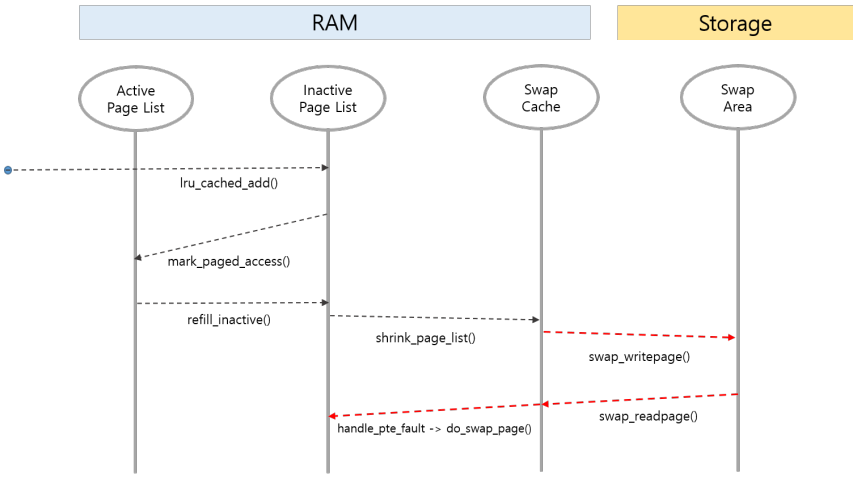
2.1 Page Reclamation Policy

Linux supports a virtual memory address which simulates a continuous physical memory for applications [10]. They may be physically fragmented and overflowing into disk storage. The Linux virtual memory system maps the virtual address space in the application program into page frames. As a technique for implementing the virtual memory, the system uses a demand paging scheme that copies only attempts to access disk pages (i.e., if a page error occurs) in the physical memory.

When the system runs out of memory, the kernel performs page frame reclamation. The page frame recovery procedure selects the victim page in a LRU (Least Recently Used) or pseudo-LRU manner. Once the file-backed page becomes dirty, the kernel writes the contents of the page frame to that disk file. If the page is an anonymous page, which is not backed with a disk file, the kernel stores the page to the swap area. Each swap area consists of 4K byte blocks



(a) Linux Reclamation Overview



(b) Kernel Control Flow of Swap In/Out

Figure 2.1: Execution of Swap Management

in a swap slot order and is used to include swap-out pages. The swap system manages three types of pages: anonymous page memory of the process, dirty pages of the process's private memory and pages that are shared by multiple processes.

2.2 Linux Swap Management

In the Linux OS, the swapped-out page resides in the swap area, and it has two types: either a *swapfile* or *device partition*. The swapfile supports a more flexible management compared to the device partition. On the other hand, using the device partition as a swap area could perform faster.

The swap cache resides in the RAM and acts as a buffer for the swap area. Shared pages are loaded on anonymous pages. For anonymous pages that are shared by multiple processes, pages stay in the swap-cache. When the system has insufficient memory, it scans the inactive list of pages and selects the page and moves it to the swap cache. The swap cache bridges the page management and data transfer between the memory and swap area. A page (or a memory page) is mapped and managed in the page table as a continuous multi-block of virtual memory. The virtual memory enables access to the pages in a typical manner without considering the current state of each processor. The storage performance is relatively slower than the memory; thus, instead of taking the page out right away, a small partition of memory is used to still hold the evicted page.

2.3 Linux System Calls

Linux supports system calls for high-level API() to pass the intention directly to the system. madvise() [3] is used to pass the direction to the kernel about the

requested virtual address range with a flag to improve the system or application performance. mlock() [2] locks part of the virtual address space for the calling process preventing the page from being used in the swap operation. The built-in functions builtin prefetch() from Ubuntu GCC [20] and pragma prefetch from Intel ICC [21] provide built-in functions to prefetch the page from the cache level. In compile-time, a user passes the intention of prefetching the page to the system. However, there are no functions for directly prefetching a swap out page.

Chapter 3

Design and Implementation

3.1 Design

In this section, we present the design and implementation of swpTracer. Figure 3.1 is a sketch of the swpTracer architecture which are divided into two big modules. We first describe the mechanism at the kernel level and then explain the user level modules.

3.2 Implementation

3.2.1 Kernel Level

After the user launches a task, the executor turns on Algorithm 1 of the swpTracer kernel module. The swpTracer module manages the swpLogger, which leaves a log when a kernel swap-related function is called. The kernel part of swpTracer follows the kernel control flow. Figure 2.1b depicts the data structure of the pages in the system and the related actions. The swpTracer module

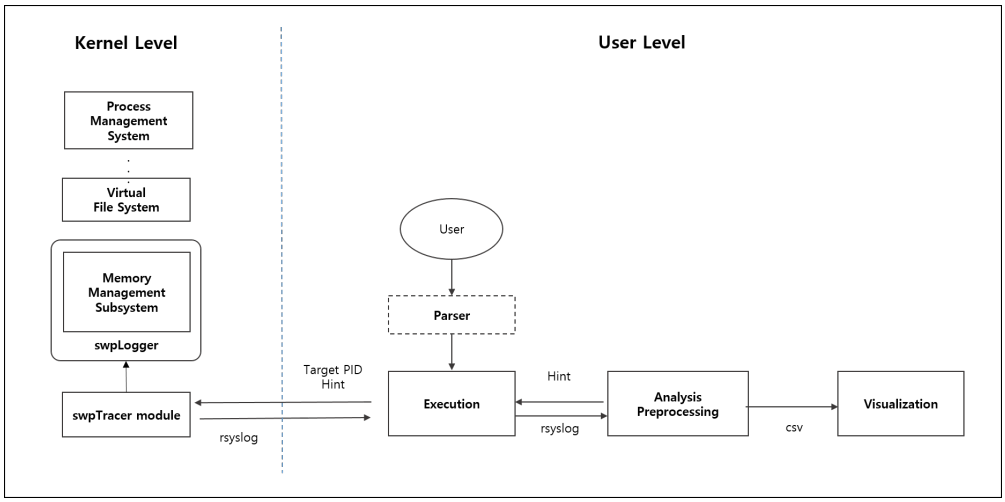


Figure 3.1: swpTracer Architecture

Algorithm 1 kernel swap module

procedure SWPTRACE($pid\_t, target$)

  if $is\_current\_pid = target\_child$ **then**

    find the vma of anonymous page

    reverse map and traverse anonymous page tree

    handle error

  end if

end procedure

controls the logger activation and checks whether the current process is derived from the target process (launched).

3.2.2 Application Level

The application level consists of three modules: executor, analyzer, and visualization. The execution module prepares the setup for the task user request and launches by passing the process identifier to the kernel.

Algorithm 2 Code Injection

procedure ($Injection(FILES)$)

  for $file$ in $FILES$ **do**

    if $line$ ==regex.matched(re) **then**

      extract data from line

      modify line from extracted data

      $newline \leftarrow modified\_line$

      write to file

    end if

  end for

end procedure

Code Injection swpTracer supports the information of the object by parsing the source code of program statically. With Flex, the injection module in Algorithm 2 traverses through the program source code statically and searches for memory allocations. The injector inserts the code to achieve several pieces of information such as the file name, function, variable name, address and requested size.

Once the analyzer module predicts whether it holds the locality, the module generates a modified code with the system call mlock() while still taking into consideration the available memory and the size of the data-object.

Analyzer and Visualization

The analyzer and visualization modules are called after the execution terminates. From the rsyslog log file, the analyzer extracts data and normalizes the value to calculate the ratio of each memory mapped region. The virtual address address logs of the swapped pages are analyzed statistically and used to determine whether they were sequential accessed and hold locality. The visualization module generates a scatter plot taking into consideration the normalized region values. Figure 3.2 shows a collection of reports generated with swpTracer applied to the benchmark Spec CPU 2006. Each plot displays the trace of the swapped in/out pages. Each subfigure contains the linear accessing interval with the swapping in/out of the pages. By combining the data object information with the swap report, we could optimize the program. An example is described in the evaluation part of this paper.

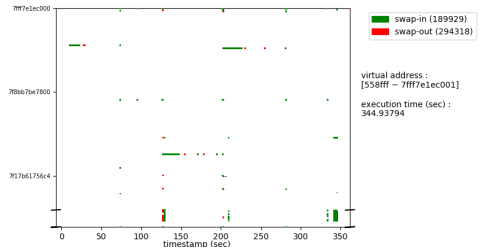
Additional: Swap-Prefetch Locking the page in memory or passing the hint to the system call did work well in a state in which the available memory is slightly insufficient, but the percentage of speed up decreases when the available memory becomes sufficient. To improve the speed up, swap-prefetch in Algorithm 3 used an additional thread to read ahead the swapped-out pages which are accessed soon afterwards. The usage of swap-prefetch is same as other prefetch built-in functions [20, 21]. The loaded chunk size will be determined by 1) the available memory, 2) the current position, 3) the total object size and 4) the starting point to launch the additional thread. Especially for the 4th condition, unnecessary page loads when a swap did not occur will decrease the performance because the swap prefetch targets swapped out pages.

Algorithm 3 Swap Prefetch Example

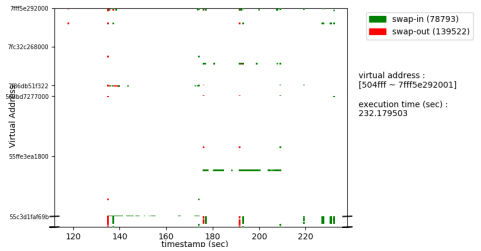
```
procedure PREFETCH(address, size, offset)
    chunk = ( size - available ) / chunk_size
    free(previous_chunk, chunk_size);
    map(next_chunk, chunk_size);
end procedure

procedure LOOP_EXAMPLE(array, size)
    for i ← 1 to N do
        for j ← 1 to N do
            if do_prefetch = True then
                thread(prefetch(array, size, [i, j]));
            end if
            /** do something **/
        end for
    end for
end procedure
```

Workloads	Descriptions	Memory (MiB)
401.bzip2	Compression	857
403.gcc	C Language Optimizing Compiler	890
429.mcf	Single-depot Vehicle Scheduling	1700
433.milc	Physics / Quantum Chromodynamics	694
445.gobmk	Artificial Intelligence	37
456.hmmer	Profile Hidden Markov Model	32
462.libquantum	Physics / Quantum Computing	104
464.h264ref	Video Compression	72

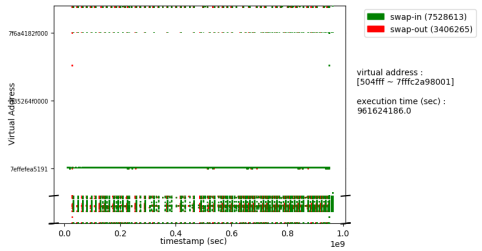
Table 3.1: Workload descriptions and maximum memory usage



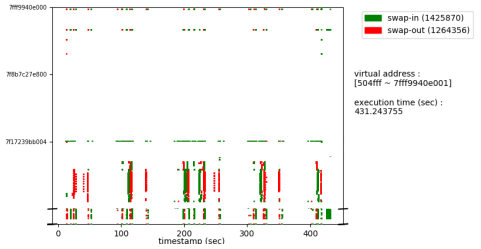
(a) Swap report of 401.bzip2



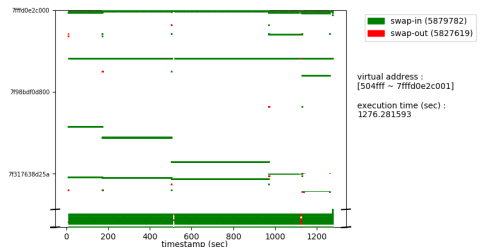
(b) Swap report of 403.gcc



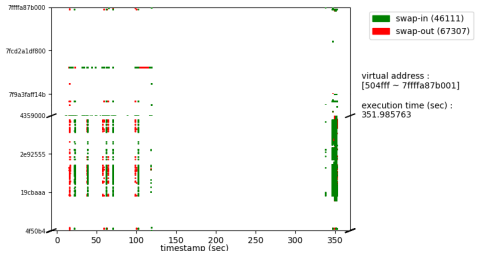
(c) Swap report of 429.mcf



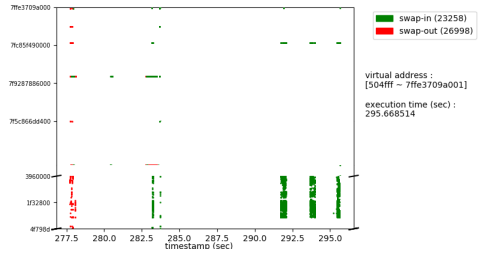
(d) Swap report of 433.milc



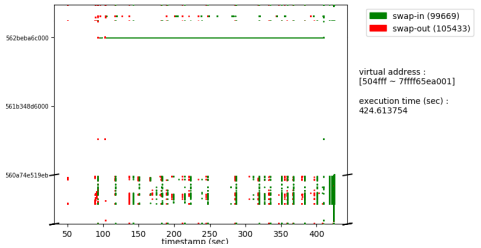
(e) Swap report of 445.gobmk



(f) Swap report of 456.hmmer



(g) Swap report of 462.libquantum



(h) Swap report of 464.h264ref

Figure 3.2: Swap report of benchmark SPEC CPU 2006 with memory pressure

Chapter 4

Evaluation

4.1 Experimental Setting

The experiment was done on a server with an Intel(R) Core (TM) i7-4790 CPU @ 3.60GHz and 32 GiB DIMM DDR3 1600 MHz (8 GiB x 4). For the swap area, the swap device partition was used with the Samsung SSD 850 (256GB, up to 5x0 MB/sec). The swpTracer supports Ubuntu and Centos as the operating system and was tested in kernel versions from 5.1.14 to 5.6.9. The application level was mostly implemented with python version 2.7.17 and other libraries or the languages are specified in the requirements list. As a simulation of a memory pressure environment, we used cgroup[12] to limit the memory usage.

4.2 Experiment

4.2.1 Generality of swpTracer

Generality is an important factor so that the tool can be used broadly. 429.mcf from Spec CPU 2006 was used to evaluate the effectiveness. mlock() was used to increase the performance, which locks the requested virtual address range in the memory. Figure 4.1 shows the result of the speed up per memory usage. The value inside the parenthesis is the percentage compared with the maximum usage. Once the system ran out of memory for 10%, the execution time increased by about 3x from 1678 seconds to 5676 seconds.

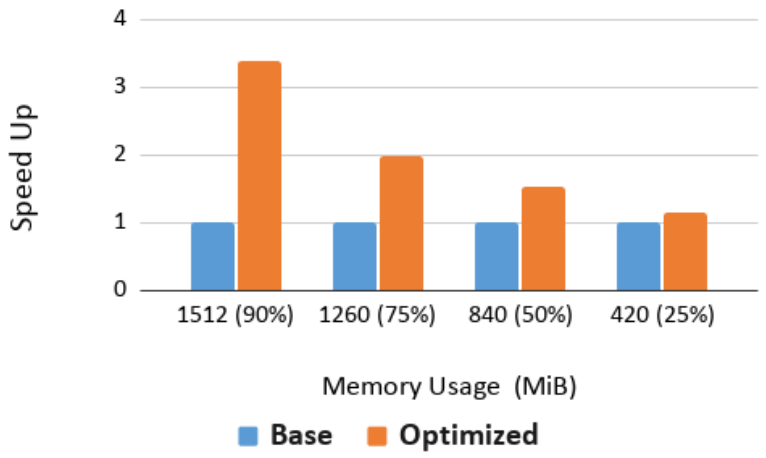


Figure 4.1: Speed up in 429.mcf

As a result of the enhancement, the application speeds up by 2x on average. Especially, when there was a 10% lack in memory, it speeds up by 3x which means that it maintains the speed same as in the full memory state.

Optimization	Descriptions	Applied Parameters
base	original code	
madvise	advice to the kernel	report full address range of data
mlock	lock part of virtual address space in RAM	lock full range of data if possible
prefetch	adapt swap-prefetch	address, chunk size, offset

Table 4.1: Optimization Types and Applied Parameters

4.2.2 Memory Optimization Method Comparison

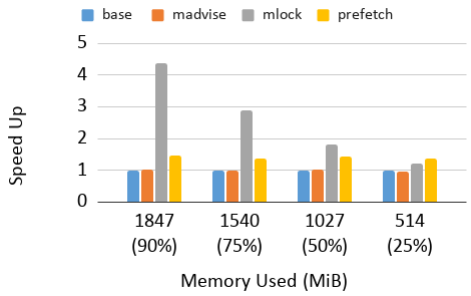
To test the effectiveness of swpTracer and swap-prefetch, we created a micro benchmark described in Algorithm 4 that traverses an array 3 times. We compared the result of the original program as the baseline and tested three cases for optimization: madvise, mlock and swap-prefetch as described in Table 4.1.

Algorithm 4 Micro-benchmark Example

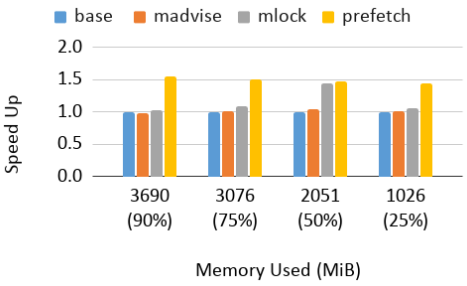
```
procedure LOOP(A[ ])
  loop ← 1
  for loop ← 1 to N do
    for k ← 1 to M do
      for k ← 1 to L do
        /* do something */
      end for
    end for
  end for
end procedure
```

We varied the size of the data arrays as 2, 4, 8, and 12 GiB. The charts in

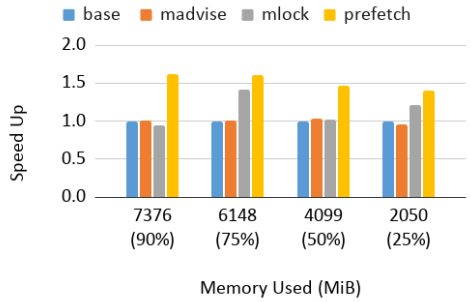
Figure 4.2 are the result of the speed up per available memory. Swap-prefetch was consistently 1.5 times faster compared to the baseline and madvise(). Relative to mlock(), it was 1.25 times faster on average. Swap prefetch was the fastest when the data size was bigger than 2 GiB while mlock showed a higher performance for a smaller object. Because mlock pins the requested range of address in the main memory, it is the fastest way to manage the memory. Once the memory ran out and the system could no longer keep the pages in the memory, swap-prefetch exceeded the performance of mlock.



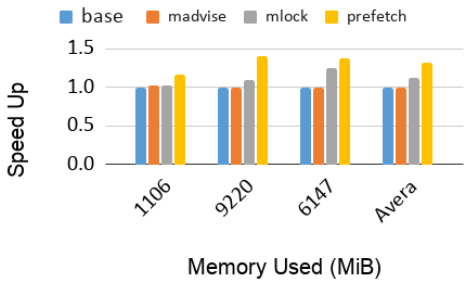
(a) Data Object Size : 2 GiB



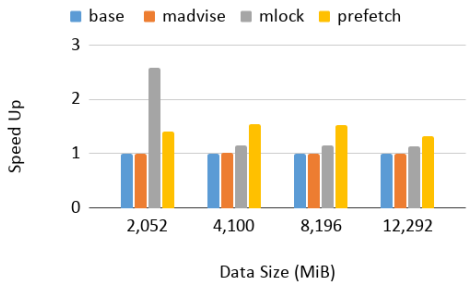
(b) Data Object Size : 4 GiB



(c) Data Object Size : 8 GiB



(d) Data Object Size : 12 GiB



(e) Average of each case

Figure 4.2: Speed Up per Available Memory (MiB)

Chapter 5

Related Work

Monitoring and analysis are combined so it is hard to separate the uses of the tool. From the point of view of system optimization, we will focus on the tool. Meanwhile, in aspect of the analysis, we want to explain the variety of approaches to the analysis.

Monitoring tools Monitoring tools for memory management are categorized into two interfaces: command line and a graphic user interface. Traditionally, tools display the utilization of resources such as the CPU, memory, and network in the command line [4, 7, 8, 11]. Lancet is used to measure the open-loop tail latency present in µs-scale data center applications with high fan-in connection patterns [25]. vmstat is mostly used to examine the swap status and display the amount of swap in/out. Each enterprise supports analysis tools for there own system architectures [5, 6].

Program Analysis and Optimization: The analysis of a program is used for system forensics and optimization of system resources. Methods include reverse-engineering and analysis along the control flow of the programs.

In reverse-engineering, hex editors are used to restore the source code. Conti et al. presented design principles for file analysis with few semantic information [13]. WinHex especially helps to interpret files reliably and accurately and recover deleted files [14, 15].

Program analysis tools such as CAMP, LLVM, and FLex are used to determine the memory access pattern [26, 34, 35, 36]. Alternatively, to determine a narrow range of targeted information, hooking a library with a user defined library is available. Hashemi et al. used the LSTM model to learn the access pattern and optimize a program [18].. Ben-Nun et al. optimized a multi-core with the memory access pattern [19].

There have been studies to that optimized the swap memory. Con Kolivas suggested the a kernel patch which manages the swapped list with a timer and prefetches the swap pages every 5 seconds within certain conditions [17]. Park et al. presented the ten- dency of memory access patterns and passed along the access patterns by code injection as a hint to im- prove the performance by code injection [34, 35, 36]. Yu et al. verified that swap could be a great solution in embedded and limited resource environments **[20]**. Choi et al. presented optimized swap mechanisms for an in-memory file system [28]. swpTracer is the a tool to that provides a novel approach in to swap management.

Chapter 6

Conclusion

Considering that the use of memory is one of the key parts of modern computer architecture, insufficient memory could be the most lethal problem for memory intensive applications. The development of hardware and software has rapidly led to areas such as big data, HPC, and machine learning. At the same time the memory usage increases along and faces a new turning point.The sharing of resources between heavy workloads in a server system results in memory shortages. Memory is one of the scarcest resources and needs to be managed.

The operating system exports some pages to storage and then loads the requested pages into memory. The delay due to the swap is one of the major problems with the overall performance degradation, given that the storage performance is slower than the memory performance. Thus, we designed and implemented swpTracer that optimizes a program by analyzing and profiling the program to prevent the swap from decreasing the performance. The swap prefetching was consistently 1.5 times faster compared to the baseline and madvise() and faster than mlock() on average by 1.25 times.

Bibliography

[1] Mandelman, J. A., Dennard, R. H., Bronner, G. B., DeBrosse, J. K., Divakaruni, R., Li, Y., Radens, C. J. (2002). Challenges and future directions for the scaling of dynamic random-access memory (DRAM). IBM Journal of Research and Development, 46(2.3), 187-212.

[2] mlock manual page, https://man7.org/linux/man-pages/man2/mlock.2.html

[3] madvise manual page, https://www.man7.org/linux/man-pages/man2/madvise.2.html

[4] atop manual page, https://www.atoptool.nl/

[5] NVIDIA Nsight Graphics manual page, https://developer.nvidia.com/nsight-graphics

[6] AMD uProf manual page, https://developer.amd.com/amd-uprof/

[7] htop - an interactive process viewer, https://htop.dev/

[8] pin manual page, https://software.intel.com/sites/landingpage/pintool/docs/97438/Pin/html/

[9] SPEC CPU 2006 manual page, https://www.spec.org/cpu2006/Docs/

[10] D. P. Bovet and M. Cesati, Understanding the Linux Kernel, 3rd Edition, O'Reilly, 2005.

[11] valgrind manual page, https://valgrind.org/

[12] cgroup manual page, https://man7.org/linux/man-pages/man7/cgroups.7.html

[13] Conti, G., Dean, E., Sinda, M., Sangster, B. (2008, September). Visual reverse engineering of binary and data files. In International Workshop on Visualization for Computer Security (pp. 1-17). Springer, Berlin, Heidelberg. https://link.springer.com/book/10.1007/978-3-540-85933-8 https://link.springer.com/book/10.1007/978-3-540-85933-8

[14] winhex - manual page https://www.x-ways.net/winhex/

[15] Tool review - WinHex https://doi.org/10.1016/j.diin.2004.04.001

[16] Zhong, K., Wang, T., Zhu, X., Long, L., Liu, D., Liu, W., ... Sha, E. H. M. (2014, October). Building high-performance smartphones via non-volatile memory: The swap approach. In 2014 International Conference on Embedded Software (EMSOFT) (pp. 1-10). IEEE., https://ieeexplore.ieee.org/abstract/document/6986137

[17] Con, K., Swap prefetch patch, http://www.users.on.net/ ckolivas/kernel/

[18] Hashemi, M., Swersky, K., Smith, J. A., Ayers, G., Litz, H., Chang, J., ... Ranganathan, P. (2018). Learning memory access patterns. arXiv preprint arXiv:1803.02329. https://arxiv.org/abs/1803.02329v1

[19] T. Ben-Nun, E. Levy, A. Barak and E. Rubin, "Memory access patterns: the missing piece of the multi-GPU puzzle," SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, TX, 2015, pp. 1-12, doi: 10.1145/2807591.2807611.

[20] GCC __builtin_prefetch() manual page, https://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html

[21] ICC prefetch manual page, https://software.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top/compiler-reference/pragmas/intel-specific-pragma-reference/prefetch-noprefetch.html

[22] Yu, X., Hughes, C. J., Satish, N., Devadas, S. (2015, December). IMP: Indirect memory prefetcher. In Proceedings of the 48th International Symposium on Microarchitecture (pp. 178-190). https://dl.acm.org/doi/pdf/10.1145/2830772.2830807

[23] Zhong, K., Zhu, X., Wang, T., Zhang, D., Luo, X., Liu, D., ... Sha, E. H. M. (2014, August). DR. Swap: energy-efficient paging for smartphones. In 2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED) (pp. 81-86). IEEE..10.1145/2627369.2627647

[24] Kwon, O., Koh, K. (2007, October). Swap-aware garbage collection for nand flash memory based embedded systems. In 7th IEEE international conference on computer and information technology (CIT 2007) (pp. 787-792). IEEE. 10.1109/CIT.2007.76

[25] Kogias, M., Mallon, S., Bugnion, E. (2019). Lancet: A self-correcting latency measuring tool. In 2019 USENIX Annual Technical Conference (USENIXATC 19) (pp. 881-896).https://www.usenix.org/conference/atc19/presentation/kogias-lancet

[26] Allen, T., Feng, X., Ge, R. (2019, May). Slate: Enabling Workload-Aware Efficient Multiprocessing for Modern GPGPUs. In 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 252-261). IEEE. , doi: 10.1145/2627369.2627647.10.1145/2627369.2627647

[27] Park, C., Kang, J. U., Park, S. Y., Kim, J. S. (2004, August). Energy-aware demand paging on NAND flash-based embedded storages. In Proceedings of the 2004 International Symposium on Low Power Electronics and Design (IEEE Cat. No. 04TH8758) (pp. 338-343). IEEE. 10.1109/CIT.2007.76

[28] J. Choi, J. Ahn, J. Kim, S. Ryu and H. Han, "In-memory file system with efficient swap support for mobile smart devices," in IEEE Transactions on Consumer Electronics, vol. 62, no. 3, pp. 275-282, August 2016, doi: 10.1109/TCE.2016.7613194.

[29] Li, L., Chapman, B. (2019, November). Compiler assisted hybrid implicit and explicit GPU memory management under unified address space. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-16). https://doi.org/10.1145/3295500.3356141

[30] Swapping and Embedded: Compression Relieves the Pressure? - Vitaly, W., Embedded Linux Conference 2016

https://openiotelc2016.sched.com/event/6DAC/swapping-and-embedded-compression-relieves-the-pressure-vitaly-wool-softprise-consulting-ou

[31] Wang, H., Zhai, J., Tang, X., Yu, B., Ma, X., Chen, W. (2018). Spindle: Informed memory access monitoring. In 2018 USENIX Annual Technical Conference (USENIXATC 18) (pp. 561-574). https://www.usenix.org/conference/atc18/presentation/wang-haojie

[32] flex manual page, http://dinosaur.compilertools.net/flex/manpage.html

[33] Gornish, E. H., Granston, E. D., Veidenbaum, A. V. (1990, June). Compiler-directed data prefetching in multiprocessors with memory hierarchies. In ACM International Conference on Supercomputing 25th Anniversary Volume (pp. 128-142).https://doi.org/10.1145/2591635.2667162

[34] Park, S., Lee, Y., Kim, M., Yeom, H. Y. (2019). Automating context-based access pattern hint injection for system performance and swap storage durability. In 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19) https://www.usenix.org/conference/hotstorage19/presentation/park

[35] Park, S., Lee, Y., Yeom, H. Y. (2019, December). Profiling Dynamic Data Access Patterns with Controlled Overhead and Quality. In Proceedings of the 20th International Middleware Conference Industrial Track (pp. 1-7) .https://doi.org/10.1145/3366626.3368125

[36] Park, S., Lee, Y., Kim, Y., Yeom, H. Y. (2019, June). Profiling Dynamic Data Access Patterns with Bounded Overhead and Accuracy. In 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W) (pp. 200-204). IEEE. https://doi.org/10.1109/FAS-W.2019.00054

초록

메모리의 사용은 현대 컴퓨터 아키텍처(폰 노이만 아키텍쳐)의 핵심 부분 중 하나이기 때문에, 메모리가 부족한 환경은 성능에 치명적인다. 하드웨어와 소프트웨어의 발전으로 빅데이터, HPC, 머신러닝과 같은 분야들이 빠른 속도로 발전하여 그에 따라 메모리의 사용량도 증가한다. 따라서, 메모리가 제한된 임베디드 환경이나, 여러 작업이 동시에 수행되는 서버에서 메모리 부족으로 작업이 중단되는 문제가 발생한다.

시스템이 메모리가 부족하면 운영 체제는 일부 페이지를 저장소로 내보낸 다음 요청된 페이지를 메모리에 로드한다. 스토리지 성능이 메모리보다 느리다는 점에서 스왑에 의한 지연은 전반적인 성능 저하의 중요한 문제 중 하나이다. 따라서 스왑이 프로그램 수행 시간에 영향을 미치지 않도록 프로그램의 스왑 발생 추이를 분석하여 스왑 발생을 줄일 수 있도록 힌트를 주는 도구인 swpTracer를 설계, 실행했다. mlock을 사용하여 Spec CPU 2006 벤치마크 중 429.mcf에 적용했을 때 기존 프로그램 대비 2, 3 배 성능이 빨라졌다.

기존의 시스템 콜을 사용하여 최적화했을 때 메모리가 살짝 부족한 경우에는 비슷한 성능을 보여주지만, 메모리가 50% 부족한 순간부터 성능 향상 폭이 줄었다. 이를 보완하기 위해 스왑 아웃 되었던 페이지를 미리 읽어두는 swap-prefetch를 구현했다. 배열을 3번 횡단하는 프로그램을 대상으로 배열의 크기를 조절하면서 swap-prefetch의 성능을 시험했다. 원본 코드와 시스템 함수인 madvise를 사용했을 때보다 평균적으로 1.5 좋아졌다. 또, swap-prefetch를 다른 시스템 함수를 사용했을 때와 mlock과 비교했을 때 평균 1.25배 성능이 빨라졌다.

주요어: Memory, Swap, Visualization, Profile, Optimization

학번: 2019-25099