Monitoring Memory Behaviors and Mitigating NUMA Effects on System with Non-Volatile Memory

Shengjie Yang[1,2], Xinyu Li[1,2], Xinglei Dou[1,2], Xiaoli Gong[3], Hao Liu[4], Li Chen[2], Lei Liu*[1,2]

[1]Sys-Inventor Lab, [2]SKLCA, ICT, CAS; [3]NKU; [4]AMS, PLA & Tsinghua

*corresponding author (PI):lei.liu@zoho.com;liulei2010@ict.ac.cn

Abstract. Non-Volatile Memory with byte-addressability invites a new paradigm to access persistent data directly. However, this paradigm brings new challenges to the Non-Uniform Memory Access (NUMA) architecture. Since data accesses cross NUMA node can incur immense performance loss, and, traditionally, kernel moves data to the NUMA node where the process accessing it locates to reduce the access latency. However, we find challenges when migrating data on NVM, which motivates us to migrate process instead. We propose SysMon-N, an OS-level sampling module, to obtain access information about NVM in low overhead. Furthermore, we propose N-Policy to utilize the data collected by SysMon-N to guide process migration. We evaluate SysMon-N and N-Policy on real-world NVM devices. The experimental results show that they provide 5.9% to 3.62x bandwidth improvement in the case where cross-node memory accesses happen.

Keywords: Non-volatile memory, NUMA, OS, Migration

1 Introduction

Non-volatile memory (NVM) attaches to the memory bus promises DRAM-like latency, byte-addressability, and data persistence. NVM will become commonplace soon. Previous studies, e.g., [4], focusing on kernel-bypassing, has redesigned the file system dedicated to NVM for reducing software overheads stemmed from kernel involvement. A critical feature of this kind of file system is the "direct access" (i.e., DAX) style interface through the *mmap*() system call, by which the user process can map the NVM-based file into its address space and access the file content directly by load/store instructions from user space. Different from the on-demand paging data access, NVM possesses both byte-addressability and persistency, which allows user processes to access the persistent data directly. However, NVM is often attached to one fixed node in modern NUMA-achitectured servers, thus it may lead to the risk of cross-node accesses (i.e., remote access). Remote access may cause dramatic performance degradation, and there are many efforts to provide shreds of evidence for this.

In terms of the performance loss due to the remote accessing on DRAM, previous work moves data from remote NUMA node to the local node where the user process is running on. However, we find some challenges from the previous studies about the NVM-based systems. (1) There is no "struct page" for persistent data in NVM that managed by the DAX-aware file systems [1], leading to

This project is supported by the National Key Research and Development Program of China under Grant No.2017YFB1001602 and the NFSC under grants No.61502452, 61902206, 61702286.

the complexity of page migration on the system using both DRAM and NVM. (2) Since the data blocks to be migrated are persistent, the process of page migration needs to be guaranteed as atomic and consistent using a transaction-like mechanism, which will introduce extra overheads on the critical path. (3) The persistent data usually has a much larger size than the volatile data, and frequent migrating of them will produce significant overheads. These challenges motivate us to seek a new design.

In this work, we explore an new mechanism. In this mechanism, instead of moving persistent data, we migrate the process to the original node where the persistent data locates. In order to achieve our goal, we then propose SysMon-N and N-Policy. SysMon-N is an OS-level memory behavior sampling module that can obtain the NVM access "hotness" (i.e., access times within a sampling interval) and the access mode (i.e., remote or local) for a user process with low overheads. N-Policy is a process migration policy designed for the user processes which use MVM. For instance, N-Policy reduces the expensive remote accesses to NVM by migrating the process to the node that is close to NVM. The experimental results show that SysMon-N and N-Policy can increase the bandwidth of read-intensive applications by 5.9% and the bandwidth of write-intensive applications by 2.71x to 3.62x when the incorrect core is allocated and remote access occurs.

2 The art of our design

2.1 SysMon-N

To tackle the problems mentioned above, we first design a practical OS-level memory behavior sampling module to capture the NVM access information. Our prior work [5] proposes SysMon as an OS-level memory behavior monitoring module. SysMon periodically checks the access bits in Page Table Entries (PTEs) to obtain the page hotness. However, merely checking PTEs can not distinguish whether the page is located in NVM or DRAM. So we went on to design SysMon-N, based on SysMon [5], to provide the physical address information of the data, it achieves two objectives: (1) Sampling pages in NVM to collect the page hotness information while avoiding sampling pages in DRAM to narrow down the sampling space; (2) Checking whether remote access occurs and collecting related data access information.

As a preprocessing step, SysMon-N first collects the NUMA topology information of the platform by scanning ACPI static resource affinity table (SRAT), where the topology information of all processors and memories are stored. By checking the `ACPI_SRAT_MEM_NON_VOLATILE` flag of the SRAT entries, SysMon-N can get the range of physical address of all NVM devices. Usually, the physical locations of all pages in a Virtual Memory Aera (VMA) are the same. For a specific VMA, SysMon-N gets the physical address of VMA's start page and checks whether it falls in the physical address range of an NVM device. If so, it means that all pages of the VMA are on one specific NVM device, and it is necessary to traverse the VMA's memory address. Otherwise, the VMA is not in NVM and can be skipped to narrow down the sampling space.

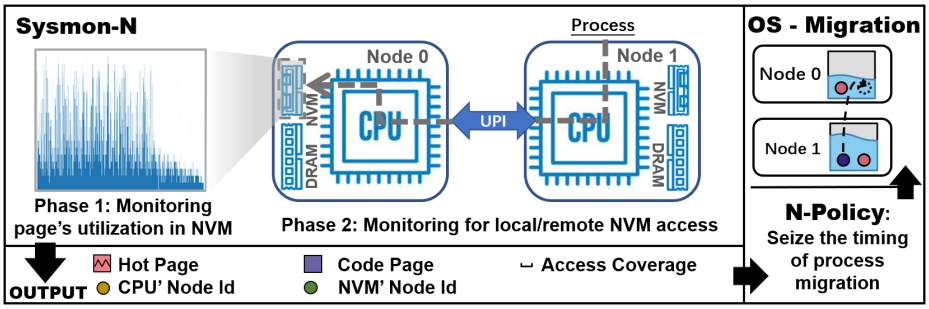


Fig. 1: Workflow of SysMon-N

Figure 1 shows the workflow of SysMon-N. It has two phases. In phase 1, SysMon-N checks each page's `access_bit` within the monitored process to find the hot pages in NVM and their corresponding physical address area. Besides, considering the massive pressure that NVM's large capacity puts on the limited number of TLBs, it is natural for OS to use the huge page on NVM. The detection of the huge page utilization in NVM are basically the same with the 4KiB-based pages, SysMon-N utilizes the PMD entries to complete address translation since the OS omits the last level PTE for 2MiB huge pages.

In phase 2, by comparing the *node id* of CPU where the process running on and that of the NVM node where the data are stored, SysMon-N can determine whether remote access has occurred or not. SysMon-N first obtains the set of CPUs on which it's eligible to run by checking process's CPU affinity mask, and then calls the *cpu_to_node*() kernel function to check the node corresponding to the CPU. Finally, SysMon-N compared the CPU node id and NVM node id to get the result: if the two node ids are the same, the process has accessed the page on remote NVM; otherwise, the process only touches local NVM.

Finally, after sampling, SysMon-N will count the number of hot and cold pages and related physical address ranges, and provide the information to N-Policy for decision-making.

2.2 N-Policy

N-Policy leverages the formation provided by SysMon-N, and guides process migration according to mitigate remote access. The key component of N-Policy is a *conditional migration model* which is depicted in figure 2. It bases on two

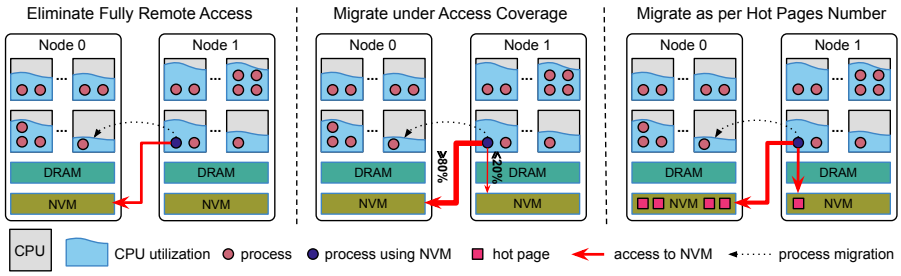


Fig. 2: Conditional Migration Model of N-Policy

principles: (1) Eliminating remote access whenever possible; (2) Trying to avoid unnecessary migration. The input of N-Policy from SysMon-N is as follows: (1) The number of hot and cold pages on per node; (2) Access coverage for each node; (3) ID of the node where CPU used by the process is located; (4) ID of the node where NVM used by the process is located.

After each SysMon-N sampling epoch is completed, the N-Policy immediately decides whether to migrate based on three data access conditions as shown in figure 2. The first situation is when completely remote access occurs (no data are accessed from local NUMA node), the migration action is triggered to migrate the process to the same node of data. This situation is determined by judging whether there is intersection between the set of the node id of CPU which the process are allowed to run on and that of NVM where accessed data locates. N-Policy makes process migration in this case, easing the overhead of cross-node access by placing the process on the same node as the NVM being used.

Further, if the two set of node ids have intersections, N-Policy compare *access coverage* on different nodes to further decide whether to execute migration. Access coverage symbolizes the amount of data accessed by the process on each NUMA node. If the access coverage of different nodes is unbalanced (i.e., N-Policy consider access coverage on remote NUMA node greater than 80% as unbalanced access). N-Policy will select the least utilized CPU on the NUMA node with the broadest access coverage as the target of process migration.

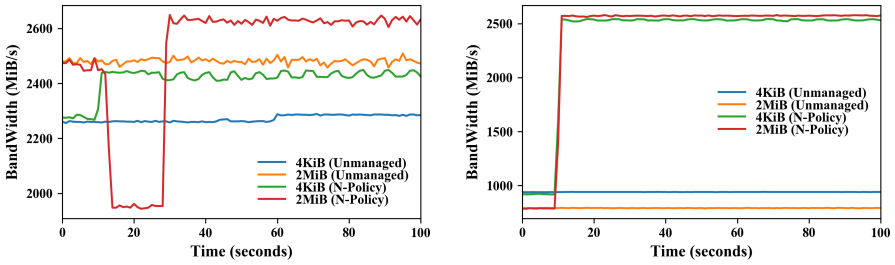
Finally, information about page hotness is also taken into account in our design. Hot pages mean frequently accessed pages and the data in them is relatively more important than other pages, and should be accessed closer in the term of latency. To ensure fast access of hot data, under the condition of node access coverage balance, N-Policy compares the hotness of each page of each NUMA node, and migrate the process to the node with the largest number of hot pages.

To avoid much overhead caused by repeated and meaningless migration. We only set N-Policy to receive messages from SysMon-N every 10 seconds. N-Policy uses the function *sched_getaffinity*() of the Linux kernel to bind a process to the corresponding CPU nodes for migration.

3 Effectiveness of N-Policy on Bandwidth and Latency

Our experimental platform is a server with dual CPU sockets of Intel Xeon Gold 6240M CPU (each has 36 cores); it has 512GB Intel® Optane™ DC persistent memory on per socket, i.e., 1024GB NVM on our platform. We first configure the namespace [3] for the Optane PMM, which represents a certain amount of NVM that can be formatted as logical blocks, and then deploy the ext4-DAX file system on it to support direct data access.

We use the Flexible I/O Tester (Fio) [2] with `libpmem` engine to evaluate the effectiveness of N-policy collaborated SysMon-N. We adjust the minimum read/write block size of I/O operations to perform reads and writes to NVM in different situations, and record bandwidth under different block sizes with and without N-Policy enabled, respectively. To verify the effectiveness of N-Policy, all data accesses of Fio are set as remote access.



(a) 100% read (b) 100% write

Fig. 3: Unmanaged vs. Use N-Policy to guide migration

Figure 3-(a) presents the 100% read case. As a baseline, we launch Fio in two cases with a single data access size of 4KiB and 2MiB, respectively. The corresponding memory bandwidth of the two cases is stable with an average of 2273 MiB/s and 2482 MiB/s. When N-Policy is enabled, it conducts process migration to eliminate the remote access which occurs at the timing around 10s in figure 3. The bandwidth changes accordingly as the process is migrated to the optimal node. N-Policy can improve the bandwidth by 6.94% and 5.90% for 4KiB and 2MiB block size, respectively. Figure 3-(b) shows the 100% write case. N-Policy achieves better results in this case. By eliminating the remote access with process migration, the bandwidth of Fio can increase by 2.71x and 3.26x in the case of 4KiB and 2MiB block sizes, respectively. This is because reading and writing bandwidth on NVM are not symmetric and NVM is more sensitive to write operations.

References

1. "Direct Access for files", `https://www.kernel.org/doc/Documentation/filesystems/dax.txt`.
2. "Fio - Flexible I/O tester", `https://fio.readthedocs.io/en/latest/`.
3. "Persistent Memory Concepts", `https://docs.pmem.io/ndctl-user-guide/concepts`.
4. D.S.Rao, et al, "System software for persistent memory", in EuroSys, 2014.
5. M.Xie, et al, "Sysmon: Monitoring memory behaviorsvia OS approach", in APPT, 2017.
6. L.Liu, et al, "Hierarchical Hybrid Memory Management in OS for Tiered Memory Systems", IEEE Trans. Parallel Distrib. Syst. 30(10), 2223–2236, 2019.
7. S.Chen, et al, "Efficient GPU NVRAM persistence with helper warps", in DAC, 2019.
8. X.Li, et al, "Thinking about A New Mechanism for Huge Page Management", in APSys, 2019.
9. F.Lv, et al, "Dynamic I/O-Aware Scheduling for Batch-Mode Applications on Chip Multiprocessor Systems of Cluster Platforms", J. Comput. Sci. Technol. 29(1), 21–37, 2014.
10. L.Liu, et al, "Going vertical in memory management: Handling multiplicity by multi-policy", in ISCA, 2014.
11. L.Liu, et al, "BPM/BPM+: Software-based dynamic memory partitioning mechanisms for mitigating DRAM bank-/channel-level interferences in multicore systems", TACO, 11(1), 5:1-5:28, 2014.
12. L.Liu, et al, "A software memory partition approach for eliminating bank-level interference in multicore systems", in PACT, 2012.
13. L.Liu, et al, "Rethinking Memory Management in Modern Operating System: Horizontal, Vertical or Random?", IEEE Trans. Computers, 65(6), 1921–1935, 2016.