Contribution ID: **111** Type: **not specified**

Tracing Data Access Pattern with Bounded Overhead and Best-effort Accuracy

Tuesday, 10 September 2019 15:00 (45 minutes)

Background

Memory pressure is inevitable in many environments. A decade size survey[1] of DRAM to CPU ratio in virtual machines and physical machines for data centers implies that the pressure will be even more common and severe. As an answer to this problem, heterogeneous memory systems utilizing recently evolved memory devices such as non-volatile memory along with the DRAM are rising.

Nevertheless, because such devices are not only denser and cheaper but also obviously slower than DRAM, more optimal memory management is required.
For this reason, various novel approaches[2,3,4] have proposed and discussed.
One common goal of general memory management mechanisms including such approaches is placing each data object in proper location according to its data access pattern. Thus, knowing the data access pattern of given workloads is key.

The Linux kernel is utilizing pseudo-LRU scheme for the purpose but it sacrificed too much accuracy for low overhead. Some approaches[3,5,6] track the access pattern based on page table access bit but such a technique could incur arbitrarily high overhead[3,6] or low accuracy[5] as the size of target workloads grows.

Our solution

We are developing a data access pattern profiling technique and tools that allow users to control the upper bound of profiling overhead while providing a best-effort quality of the result regardless of the size of the target workload.
Basically, the solution is implemented on page table access bit sampling, which is widely used from other approaches[5]. In this approach, users can control the upper bound of the profiling overhead by setting the total number of the sampling regions.

What differentiates ours from the others is its adaptive classification of sampling regions. The algorithm adaptively merges and splits each sampling region so that every data item in each region to have a similar data access pattern. In this way, our mechanism can minimize the number of sampling regions while maximizing the profiling accuracy.

Implementation

We implemented the mechanism as a kernel module that interacts with userspace via the 'debugfs' interface. We also provide userspace tools that help the use of the interface and visualization of the profiled results.

Expected users

We believe our mechanism could be used by both kernel space and user space.

In kernel space, the aforementioned heterogeneous memory management approaches[2,3] could directly use this mechanism for efficient and effective data access pattern exploitation. Furthermore, this can be used

by many traditional memory management subsystems that relying on the kernel's pseudo-LRU or naive assumptions. For example, selection of victim pages for page cache eviction or swap, pages to be promoted or demoted to or from huge pages (THP), target pages to compact nearby could use this.

In userspace, system administrators or application programmers could use this tool to quickly analyze their workloads. The result can be used for a various way. Administrators might use the result to know a performance-effective working set size of their workloads and operate their system more efficiently.
Programmers would optimize their programs using madvise()-like system calls[5] to give data access pattern hints to the system.

Evaluations

We applied this resulting tools to more than twenty of various realistic workloads including scientific, machine learning, and big data applications and confirmed that it provides effective and efficient profiling. For the confirmation, we visualized the output and compared with manual code review. We also evaluated its usefulness by manually estimating the performance-effective working set size and optimizing with madvise() system calls. Our performance-effective working set size was similar to that we found using time-consuming repetitive experiments and the optimization improved the performance under memory pressure situation up to 2x.

Future plans

We are planning to open source this tool and submit the patchset to LKML for upstream merge.

Expected results of this talk

We hope this talk to help discussions about the effective and efficient way to get data access pattern and how to use the data from memory management systems.
Also, we would like to hear back kernel core developers' comments for upstreaming of this tool.

References

[1] Nitu, Vlad, et al. "Welcome to zombieland: practical and energy-efficient memory disaggregation in a datacenter." Proceedings of the Thirteenth EuroSys Conference. ACM, 2018.
[2] "NUMA nodes for persistent-memory management." https://lwn.net/Articles/787418/
[3] "Proactively reclaiming idle memory." https://lwn.net/Articles/787611/
[4] "[RFCv2 0/6] introduce memory hinting API for external process." https://lore.kernel.org/lkml/20190531064313.193437-1-minchan@kernel.org/T/#u
[5] "Cache Modeling and Optimization using Miniature Simulations." https://www.usenix.org/conference/atc17/technical-sessions/presentation/waldspurger
[6] "Idle Page Tracking." https://www.kernel.org/doc/html/latest/admin-guide/mm/idle_page_tracking.html

I agree to abide by the anti-harassment policy

Yes

I confirm that I am already registered for LPC 2019

Primary authors: PARK, SeongJae; Mr LEE, Yunjae (Seoul National University); Prof. YEOM, Heon Y. (Seoul National University)

Presenter: PARK, SeongJae

Session Classification: Kernel Summit Track

Track Classification: Kernel Summit talk