AOS Exam 2019
Paper 2

Summary

This paper aims to tackle the problem of low utilization in huge pages and the problem of memory bloating as a result, especially in a cloud computing environment. Huge pages are used to reduce the number of TLB misses when a program uses a lot of memory and the memory is contiguous. To address the issues the author proposes SysMon-H – an OS-level monitoring tool used to monitor the utilization of huge pages. The author also proposes H-Policy – a new memory management system which will dynamically, based on the monitoring result of SysMon-H, split huge pages into smaller ones or vice versa. Existing research efforts into monitoring huge page usage is done by checking the "access_bit" in the page table entries. The author claims that this is suboptimal since sampling a memory footprint of 20GB takes around 6 seconds. SysMon-H offers a new approach by measuring the TLB misses, with the core idea being that pages that are accessed more often (hot pages) will have more TLB misses, and fewer for cold pages. The author presents result of SysMon-H monitoring 4 applications and shows the distribution of TLB misses within the memory layout of each application. One concern that the author raises is that there are"very hot" pages which are kept in the TLB most of the time due to frequent accesses, and might be classified as cold pages instead. The reported result is that only 0.44% of pages that are classified as cold are actually very hot.

SysMon-H is split into 2 steps: The first is to monitor the number of TLB misses as described, and the second stage is to check the access bits for the cold region to find the "very hot" pages. There is a H-Tree for each application, which is basically a binary sort tree, and "pages are organized according to their access frequency for each application", with a value "high_tlb" as the root.

The per-process H-Tree will provide information for H-Policy, which will split huge pages into smaller 4KB pages when the allocated memory amount achieves 90%. It first chooses the application with the lowest overall TLB misses and then split the huge page with the lowest number of TLB misses within that application. The second step of the H-Policy is that it will "promote" a consecutive 2MB memory space to a huge page, if 90% of 4KB pages in a contiguous 2MB space have TLB misses. This promotion step is performed every 30 seconds.

Pros

- The idea that huge pages and small pages should be used dynamically based on page utilization

Cons

- There is no clear results of how the combination of these two designs have improved existing systems other than in vague terms.
- There is little to no benchmarking (See benchmarking crimes below).
- Lack of any implementation details into the design, especially the H-Tree and H-Policy implementations
- Lack of reproducibility
- Arbitrary definition of hot and cold pages, as well as arbitrary "watermarks" constants.

These concerns are addressed in the criticism section below.

Criticism

The core idea that hot pages will have more TLB misses is not very sound. The claim that every TLB entries have the being swapped out is based on a "hash function based replacement algorithm". However they did not address that the pages that are being swapped out will necessarily have to clear their TLB entry. So the claim that the distribution of TLB entries being swapped out is uniform is untrue in a large working set when pages are being paged out often. Under the clock algorithm, page accessed most infrequently will have relatively higher TLB misses, thus their claim that higher TLB misses implies a hot page is not necessarily true in all cases.

The classification of a page as "very hot" is defined to be a page that was "touched in 3 consecutive scan interval of 1 second", and "cold" as having fewer than 10 TLB miss count in a sampling period of 5 seconds. (The definition is later contradicted by the author as having the page's access bit be 1 only twice in 3 conseutive 1-second interval scan). The labelling of hot and cold pages seems completely arbitrary, and the scanning interval of 1 and 5 seconds seem random as well. For example, a program that periodically checks one part of the memory region every 1 second (a mail server checking for new mail for instance) will have a page being labelled as very hot, when in fact it is cold since it's only accessed once per second. The author also didn't provide any details on what applications are being scanned, as well as the overall system at the time, whether there are multiple processes running at the same time so the result of 0.44% has no validity. The different watermark constants are claimed to be "empirical values based on analysis of programs" but no further information on how they are derived, how statistically significant or optimal these numbers are.

The paper claims that their design is superior to existing solution of checking "access_bits" since walking through the page table can be slow. However, the cold pages will have their "access_bits" checked to confirm that they aren't in fact hot pages. However, in the worst case scenario, every single page can be cold and thus must be checked. This is worse than simply checking all the access bit in the beginning because of the overhead of checking TLB as well as the extra data structure that was involved. Claiming that this method is only 0.1% overhead as opposed to 2.3% overhead is misleading and incomprehensive.

In H-Policy, it is not clear how the application with the lowest overall TLB misses is chosen, as the H-Tree is per application. Further, there is a clear logical flaw: that a new process spawned will clearly have the least TLB misses compared to other long running processes and will have its huge page split up although it might have achieved good utilisation of the huge page. It is also unclear how the promotion of pages is implemented.

It is claimed that "H-Policy will aggressively create large physical memory blocks by merging the adjacent small page blocks". This sounds like the existing (and old) idea of having pages coalescing. Although it claims H-Policy is "orthorgonal" to the "buddy system" in Linux, there is no real explanation of the differences. The only explanation is "tracking down holes" using counters, which to me is extremely vague and unclear of what they are doing. The paper claims there are interfaces to change the watermarks and other parameters in the prototype, but it is not elaborated how that is achieved, and potential consequences of such an interface on performance and security.

Benchmarking crimes

In section 3.3 "The Sampling Overheads" the author discusses the potential overheads of SysMon-H. We discuss the few issues with the way the author is doing benchmarking in this section as well as throughout the paper.

- Micro-benchmarking

Lack of any macro-benchmarking of the entire system. Section 3.3 discusses the overheads brought by SysMon-H in isolation, and has no discussion on this process's effect on the entire system. Further, it does not evaluate any potential performance degradation

- No indication of significance of data

  There is no standard deviation given for any figures within the results. There isn't any mention of any repeated experiments.

- Lack of reproducibility

  There is no clear explanation of how various parts of the design is implemented. For example there is no mention of how the H-Tree is being stored, where the bookkeeping happens, or if there are any overheads as a result of doing so. There is also no clear information on how these benchmark experiments are conducted.

- Subsetting of data

  The author only chose the following programs to while running SysMon-H: Memcached, Redis, deepsjeng_r and mcf_r in SPEC CPU 2017, only a tiny subset of SPEC CPU. The author claims that "Redis has the largest memory footprint in our experiments", and thus the results are "representative". This is misleading.

- No definition of overheads, relative numbers only

  The figures "0.1%" and "2.3%" overhead has no meaning since it doesn't discuss what overhead actually means. Further, there are no actual figures of what these percentages represent, and there are no baseline comparison.

- No consideration of worst case execution time

  No mention of best and worst case scenario, and an arbitrarily chosen figure is used as a conclusive evidence for overhead figures.

Conclusion

Overall this is a poorly written paper. The idea of splitting huge pages up can potentially be a good idea, but this paper lacks any solid implementation details. It also lacks any significant results or data that could be of use for future researchers.