ProSwap: Period-aware Proactive Swapping to Maximize Embedded Application Performance

Dongjoo Seo[1], Biswadip Maity[1], Ping-Xiang Chen[1], Dukyoung Yun[2], Bryan Donyanavard[3], Nikil Dutt[1]
[1] University of California, Irvine, [2] Samsung Electronics, [3] San Diego State University
{dseo3, maityb, p.x.chen, dutt}@uci.edu, dukyoung.yun@samsung.com, bdonyanavard@sdsu.edu

2022 IEEE International Conference on Networking, Architecture and Storage (NAS) | 978-1-6654-5408-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/NAS55553.2022.9925330

***Abstract*—Linux prevents errors due to physical memory limits by swapping out active application memory from main memory to secondary storage. Swapping degrades application performance due to swap-in/out latency overhead. To mitigate the swapping overhead in periodic applications, we present ProSwap: a period-aware proactive and adaptive swapping policy for embedded systems. ProSwap exploits application periodic behavior to proactively swap-out rarely-used physical memory pages, creating more space for active processes. A flexible memory reclamation time-window enables adaptation to memory limitations that vary between applications. We demonstrate ProSwap's efficacy for an autonomous vehicle application scenario executing multi-application pipelines, and show that our policy achieves up to 1.26× performance gain via proactive swapping.**

***Index Terms*—Swap system, Periodic application, memory reclamation**

I. Introduction

Modern embedded systems typically execute diverse applications on a single device, concurrently, with performance constraints for each application [1]. Shared hardware resource contention between applications at runtime adds unpredictable overhead and negatively impacts application performance [2]. Even with sophisticated resource management schemes, degraded application performance can be observed from memory-pressure-induced critical errors due to complex concurrent resource usage and dynamic application updates. Thus, careful runtime management of hardware resources is required to deliver acceptable performance while meeting latency constraints.

Main memory is a key shared hardware resource under contention by concurrent applications. Memory becomes overcommitted when the virtual memory requirements of applications and operating system are no longer satisfied by the physical memory, leading to unpredictable delays. Such unpredictable delays can lead to performance degradation in end-to-end application pipelines with periodically executing applications. Figure 1 shows an embedded autonomous vehicle computational pipeline with periodically executing applications, where the swapping-in/out overhead from overcommitted memory results in performance degradation that can lead to acute slowdowns, and even fatal errors.

The Linux swap system dynamically swaps-out rarely used physical memory pages to a disk device when memory is overcommitted, and swaps-in previously swapped-out pages as needed. Swapping prevents out-of-memory (OOM) crashes

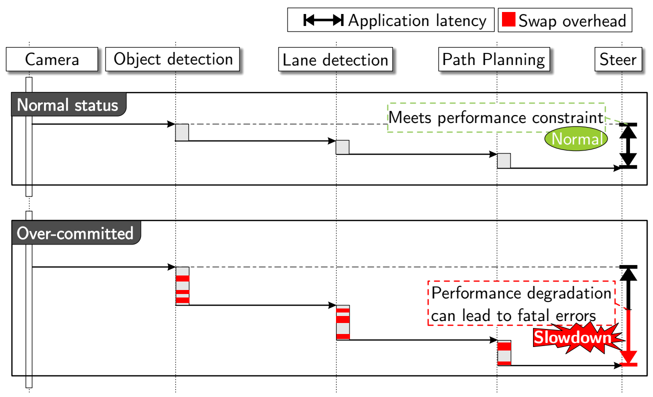

Fig. 1: Simplified end-to-end application pipeline. When memory is overcommitted, unpredictable delays from swapping can lead to acute slowdowns, and fatal errors.

at the expense of swap-in/out performance overhead. There is a rich history of research that attempts to (a) reduce the swap overhead by optimizing the swap policy, or (b) prevent physical memory overcommitment. Examples include domain-specific optimizations in mobile applications [3], DNN applications [4], and hypervisor or cloud applications [5]. These techniques are platform- or application-specific, or require modification of kernel or swap scheme. No existing techniques target periodic applications. An effective mechanism to handle overcommitted memory for periodic applications must adapt at runtime to various concurrent application sets and different degrees of memory pressure.

We propose ProSwap, an adaptive policy to proactively swap-out pages for periodic applications on embedded systems. ProSwap reduces swap penalty by proactively identifying memory pages to swap-out based on memory phasic behavior. The policy works in conjunction with the default Linux swap system without kernel modification, and can adapt to diverse concurrent applications with different periods.

The main contributions of this paper are:

- ProSwap: a *proactive* memory reclamation policy to identify memory pages to swap-out at runtime, and *adaptively* change the reclamation-window based on periodic application behavior.
- An implementation of ProSwap as a user-level swapping policy using `eBPF` (extended Berkeley Packet Filter) [6] for periodic applications that does not require source-code or kernel modifications.

- An experimental case study that demonstrates the efficacy of ProSwap for an autonomous vehicle application scenario, showing performance gains of up to 1.26× for a representative end-to-end application pipeline.

II. ProSwap: Proactive memory reclamation policy

When memory is overcommitted, memory accesses by concurrent applications frequently experience swap-in/out overhead. The overhead adversely affects the application's performance, and intensifies when many applications are running simultaneously. We make the observation that accesses to memory pages in periodic applications are amenable to proactive swapping. Accordingly, ProSwap deploys two components to reclaim memory in overcommitted scenarios: $\textcircled{1}$ a memory monitor that identifies memory pages experiencing periodic activity, and $\textcircled{2}$ a reclamation routine that proactively evicts memory pages.

A. Memory Monitor to Identify Periodic Memory Pages

Proactive eviction must target periodically-accessed memory, because reclaiming random memory pages will lead to additional overhead without improvement. Algorithm 1 specifies ProSwap's memory monitor routine that identifies periodic memory page activity. We maintain a set of virtual page numbers in each process that are swapped out along with the average time between consecutive swap-outs. We deploy a simple heuristic that defines periodic memory activity as a a page swap-out count ≥ 2, and the time-period as the average elapsed time between swaps. The memory monitor is able to update the information corresponding to memory swaps with minimal monitoring overhead by using event-driven techniques offered by `eBPF` (extended Berkeley Packet Filter) [6].

Algorithm 1 Memory Monitor routine

```
    Hash set of swap-out statistics $S : K \rightarrow V$
 1: $K$ = { Id of process, virtual page number }
 2: $V$ = { Number of swap-out, Average elapsed time }
 3: if Swap-out event occurred then
 4:     $K$ = event data
 5:     if $K$ in $S$ then
 6:         Update $V$ by event data
 7:     else
 8:         $K[S]$ = event data
 9: else
10:     Wait for event
```

B. Reclamation Routine

Algorithm 2 specifies ProSwap's reclamation routine that performs proactive eviction of periodically accessed pages to reclaim the memory before the next memory reference. We use an adaptive time window with victim memory reclamation.

Adaptive time window for reclamation: We start with a user-configurable initial time window (γ) for reclamation that adapts at runtime based on periodic activity. Note that while

Algorithm 2 Reclamation routine

```
    Sleep time multiply: $\alpha$                    ▷ configurable
    Initial timing window: $\gamma$                  ▷ configurable
    Hash set of swap-out statistics: $S$
    Timing window for reclaiming: $T$
    Application state: $A \leftarrow$ non-periodic
 1: while True do
 2:     Sort $S$ by the number of swap-outs
 3:     if number of swap out in $S[0] >= 2$ then
 4:         $A \leftarrow$ periodic
 5:         Clear pagecache
 6:         Disable readahead of disk
 7:         Disable page fault around size
 8:         $T$ = $S[0]$'s average elapsed time
 9:     else
10:         $A \leftarrow$ non-periodic
11:         Restore size of readahead and page fault arounding
12:         $T = \gamma$
13:     while During $T \times \alpha$ do
14:         if $A$ is periodic then
15:             if During $T$ and
              referenced bit of virtual page $>= 1$ then
16:                 Swap-out page
17:         else
18:             if During $T$ and
                  referenced bit of virtual page $<= 1$ then
19:                 Swap-out page
```

memory optimizations (e.g., readahead, page fault around) reduce the overhead of disk activity through localization, they create additional overhead when memory is overcommitted. We therefore temporarily disable the operating system's memory access optimizations for periodic memory accesses, and initialize the reclamation period for ProSwap (**Lines 4-8**). The reclamation period is adapted based on the average elapsed time of the application's most-swapped-out page (**Line 8**). When periodic activity is no longer detected, we restore memory access optimizations (**Lines 10-12**).

Victim memory reclaim: If the application (A) is periodic (**Lines 14-16**), we swap-out the memory pages that were already referenced in the current reclamation window, since they will not be referenced again. If A is non-periodic (**Lines 17-19**), we swap-out the memory pages that were rarely referenced by the application (number of references ≤ 1). We observe that rarely referenced pages are typically used during application warm-up and not referenced throughout runtime.

In summary, ProSwap adapts its reclamation time-window based on application phasic behavior, and uses a combination of the monitoring and reclamation routines to proactively swap-out less-referenced pages.

III. Results

We evaluate ProSwap in three execution scenarios: 1) a synthetic periodic workload, 2) exemplar autonomous vehicle

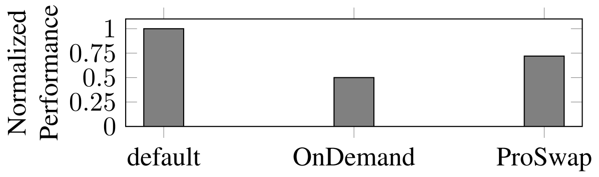

Fig. 2: Synthethic workload performace with overcommitted memory for OnDemand and ProSwap policies, normalized to non-overcommitted scenario (default)

applications in isolation, and 3) autonomous vehicle applications in end-to-end pipelines:

1) Synthetic workload: Simple memory allocation with periodic memory references.
2) Isolated representative applications: lane detection [7], path planning [8], and object detection [9]. Their periodic behavior reflects real-world autonomous vehicle software stack where the same computation is performed over repeated inputs.
3) Pipelined application set: pipeline workload with four periodic applications [8].

A. Experimental Setup

Our experimental hardware platform has an Intel i7-11700 CPU with 16GB main memory and a Crucial CT1024MX 1TB SSD. We use Linux version 5.15 and docker resource virtualization based on cgroup [10]. We emulate resource restrictions using virtual cpus (set to four cores with `--cpuset-cpus`, `--cpus` flags) and virtual main-memory (with `--memory` flag). We use `eBPF` for monitoring the memory page information from the Linux kernel and Linux `DAMON` for dropping the memory (reclamation).

B. Synthetic workload

To show the effectiveness of ProSwap, we use the memory access workload in the memory access simulator `MASIM` [11]. The workload allocates 200MB memory during initialization and references the upper 100MB and lower 100MB regions of virtual memory in an interleaved fashion.

Figure 2 shows the performance of synthetic workloads for three cases (X-axis): (1) default (no overcommitted memory), (2) overcommitted memory with OnDemand[1] swap, and (3) overcommitted memory with ProSwap. To emulate overcommitted memory, we restrict the physical memory size to 100MB when launching the container. The Y-axis shows performance as the number of memory references per second normalized to case (1). We make two key observations. First, the performance of OnDemand (normalized to default case) is 0.5, meaning the performance is directly proportional to the available physical memory (100MB instead of 200MB). Second, ProSwap improves application performance by 44% (0.72 instead of 0.5) compared to OnDemand. The performance enhancement comes from the reduction in swap-out delays. We conclude that proactive swap-out is effective for a synthetic periodic application. We note, however, that the synthetic workload tries to touch a large region of allocated memory, whereas real-world applications touch a restricted small memory region with diverse access patterns.

[1]OnDemand swapping is the default swap system in Linux.

Applications	Description
Lane Detection	Detect lane by using camera input
Path Planner	Find the path for vehicle by using inputs
Object Detection	Detect object by using camera input
Steer (DASM)	Drivers assistance system module to steer car

TABLE I: Application set from Chauffeur

C. Representative embedded applications

As an exemplar of real pipelined periodic applications for autonomous vehicles, we use selected applications from the Chauffeur benchmark suite [8] shown in Table I.

Isolated: First, we evaluate the memory footprint reduction of each Chauffeur autonomous driving application in isolation without overcommitted memory. The result is shown in Table II. We observe that ProSwap can reduce the average resident memory size dramatically through proactive memory reclamation. On average we reduce resident memory by 44%, and up to 89% for object Detection. Thus, ProSwap is effective in reducing memory requirements without sacrificing performance when applications are considered in isolation by reclaiming unused pages at runtime.

Application name	normal resident memory size	reduced resident memory size	reduction
Lane Detection	286MB	120MB	-58%
Path Planner	949MB	742MB	-21%
Object Detection	801MB	74MB	-89%
DASM	1MB	0.92MB	-8%

TABLE II: Application resident memory reduction during running phase with ProSwap

Pipelined: Next, we evaluate the performance of the entire Chauffeur autonomous driving application pipeline. Figure 3 shows the performance of the end-to-end application pipeline for different overcommitted memory scenarios. We run each experiment for five minutes and take an average across ten runs. The X-axis corresponds to different memory restrictions: Low restriction (30%), Medium restriction (50%), High restriction (70%). The Y-axis shows how many input frames are processed every second with ProSwap (black), and OnDemand (gray). We make the following observations: Performance, i.e., frames-per-second (fps), decreases when memory pressure is increased for both OnDemand and ProSwap policies. The performance decreases as memory pressure increases due to

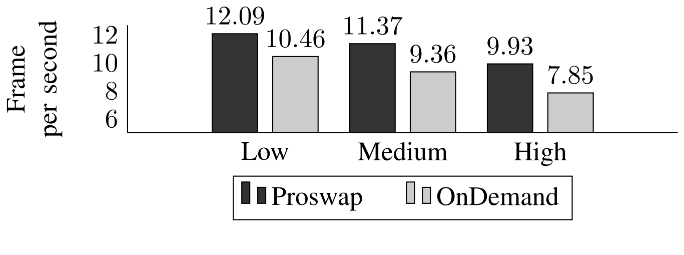

Fig. 3: Comparison of pipeline performance

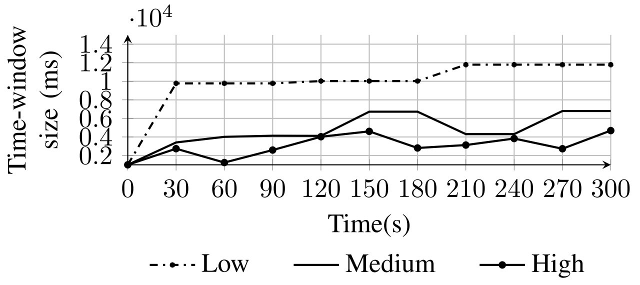

Fig. 4: Adaptive reclamation window size over time for low, medium, and high memory pressure

exacerbated swap activity. In all cases, ProSwap can increase performance over OnDemand: 10.46→12.09 (15%) for low, 9.36→11.37 (21%) for medium, and 7.85→9.93 (26%) for high memory pressure. This demonstrates the efficacy of ProSwap for a real-world application pipeline under various degrees of memory pressure. The average performance improvement in the real-world applications (up to 21%) is less than the synthetic applications (44%). This is because the synthetic workload assumes 100% memory activity with no compute cycles. This results in a proliferated gain for the synthetic workload, but does not reflect real-world behavior.

We investigate how ProSwap adapts the reclamation window in Figure 4. For low memory pressure, the reclamation window changes infrequently but in large steps due to limited swap activity. For medium and high memory pressure scenarios, the reclamation window changes soon after launching the pipeline due to high swap activity. ProSwap can necessarily adapt its reclamation window according to application.

We conclude that ProSwap can adapt to different memory pressures without any additional information from the application and proactively reclaim memory for improved performance in overcommitted memory scenarios.

IV. Related work

When a memory access causes a page fault, the page fault handler fetches the corresponding page from disk to main memory [12]. If physical memory is running out, the Linux OnDemand swap system reclaims memory by maintaining a list of inactive pages. Prior work has improved OnDemand swap by optimizing page size, queue length, and timing properties, and proposed application-specific enhancements. Kim et al. [3] suggest an application-behavior-based swap system for Android. Ko [5] and Xue [4] improve the performance by tweaking the size of each swap-out page. Park et al. [13] use object priority to guide the swap system to improve performance. However, it requires custom application-/OS-level modifications, and is not applicable for concurrently running applications.

Our ProSwap policy is the first to proactively complement the swap system by predicting periodic behavior to maximize performance when memory becomes overcommitted.

V. Conclusion

We presented ProSwap, a proactive memory reclaim policy for periodic embedded applications. We evaluated ProSwap for various applications sets including synthetic, single applications, as well as realistic end-to-end pipelined applications through an exemplar autonomous driving benchmark suite. ProSwap achieves up to 26% performance gain by proactively swapping out memory pages for periodic memory access applications, and dramatically decreases up to 89% of resident physical memory size of these applications with minimal overhead.

Acknowledgment

This work was partially supported by NSF grant CCF-1704859, a research gift from Meta Platforms, Inc., and a grant from Samsung Electronics Co., Ltd.

References

[1] A. *Malinowski* and H. *Yu*, "Comparison of embedded system design for industrial applications," *IEEE transactions on industrial informatics*, pp. 244–254, 2011.

[2] G. *Xie et al.*, "High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems," *Journal of Systems Architecture*, pp. 3–14, 2016.

[3] S. *Kim et al.*, "Application-aware swapping for mobile systems," *ACM Transactions on Embedded Computing Systems (TECS)*, pp. 1–19, 2017.

[4] F. *Xue et al.*, "Edgeld: Locally distributed deep learning inference on edge device clusters," in *2020 IEEE HPCC/SmartCity/DSS*, 2020, pp. 613–619.

[5] S. *Ko et al.*, "A new linux swap system for flash memory storage devices," in *2008 IEEE International Conference on Computational Sciences and Its Applications*, 2008, pp. 151–156.

[6] D. *Renzo* and D. *Michele*, "Berkeley packet filter: theory, practice and perspectives."

[7] A. A. Assidiq, O. O. Khalifa, M. R. Islam, and S. Khan, "Real time lane detection for autonomous vehicles," in *2008 International Conference on Computer and Communication Engineering*. IEEE, 2008, pp. 82–88.

[8] B. *Maity et al.*, "Chauffeur: Benchmark suite for design and end-to-end analysis of self-driving vehicles on embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, pp. 1–22, 2021.

[9] M. Bjelonic, "Yolo ros: Real-time object detection for ros," *URL: https://github.com/leggedrobotics/darknet_ros*, 2018.

[10] J. *Turnbull*, *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[11] S. *Park et-al.*, "Memory access simulator," https://github.com/sjp38/masim, 2021.

[12] R. V. *Riel*, "Page replacement in linux 2.4 memory management." in *USENIX Annual Technical Conference, FREENIX Track*, 2001, pp. 165–172.

[13] S. *Park et al.*, "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)*, 2019.