Early Detection and Proactive Taming of Memory Pressure Problems

Pranjal Chakraborty

Supervisor: Dr. Naser Ezzati-Jivan

Email:
pcborty@outlook.com
pchakraborty@brocku.ca

Department Of Computer Science
Brock University, ON, Canada

Introduction

- Memory pressure
    - Process(es) needing wait time to swap in pages from page cache
- Out of Memory (OOM) and OOM Killer
    - A memory stall preventing mechanism
- Application context
    - Any kind of resource constrained systems

Challenges

❑ Finding out the early signs of memory pressure

❑ Optimizing process level metric or trace data collection

❑ A better OOM killer

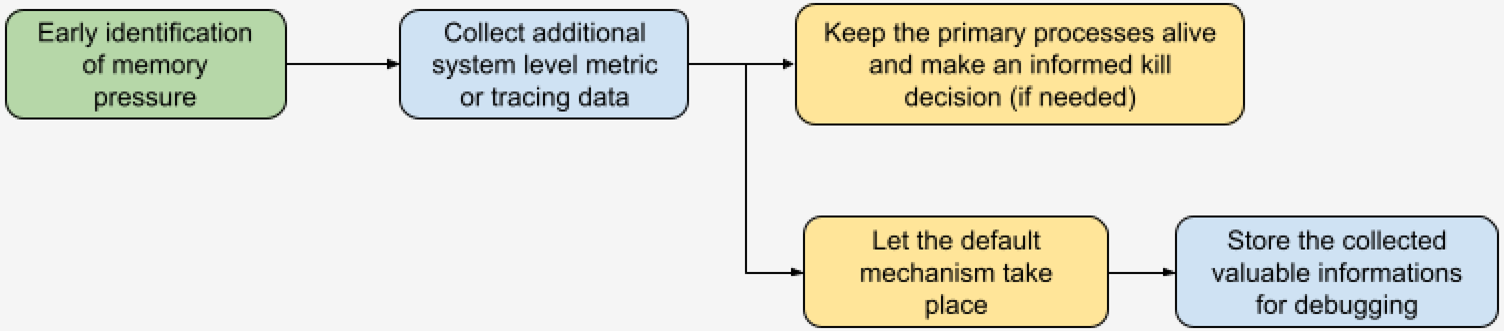
Methods to Deal with Memory Pressure

Reactive approach: The system has already experienced high memory usage and we are scrambling for any information to make a decision

- ❑ How most of the recent literature and the default OOM killer works
    - ❑ By monitoring process level metrics and triggering OOM reactively, using a new algorithm [1]
    - ❑ By exploring and comparing different configurations of `swappiness` in cgroups [2]
    - ❑ By using a novel memory pressure calculation strategy [3]

Methods to Deal with Memory Pressure

- Proactive approach



Early identification of memory pressure

Collect additional system level metric or tracing data

Keep the primary processes alive and make an informed kill decision (if needed)

Let the default mechanism take place

Store the collected valuable informations for debugging

❑ Identifying early, collect more targeted information, and make a more informed decision

❑ Little overhead, higher observability

Phase 1: Early Identification of Memory Pressure

Subproblems

- ❑ What metrics should we use and what will be the frequency?
- ❑ Can we learn the pattern of memory activities overtime and do this intelligently?
- ❑ What is the overhead of this kind of approach?

Phase 1: Early Identification of Memory Pressure

Subproblem 1.1 : Used metrics

❑ 30 memory related statistics (paging, hugepage utilization, swap page utilization, and general memory utilization)

❑ 2 kernel events call count (`mm_page_alloc`, `kmalloc`)

❑ 1 system call count (`sys_brk`)

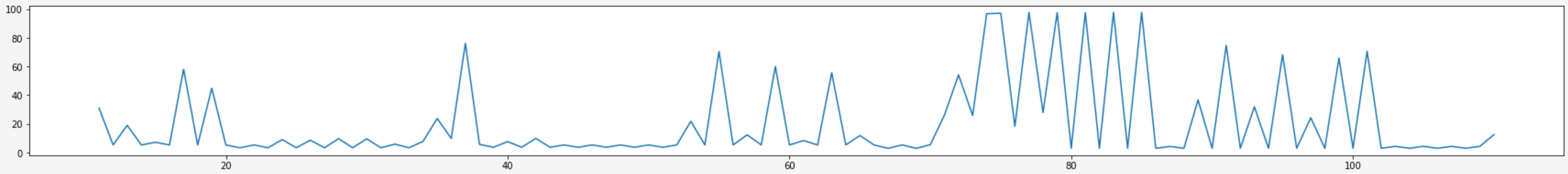
Subproblem 1.2 : Frequency

❑ 1s (500ms has higher footprint, but the performance improvement is minimal)

Phase 1: Early Identification of Memory Pressure

Subproblem 2 : A self-supervised approach to detect early signs of memory pressure

- ❑ Used benchmarking tools to create random memory activities
- ❑ Collected data for over 85000 timestamps (around 24 hours)
- ❑ Converted memory usage info into one-hot encoded phases, and used it as our dependent variable



Phase 1: Early Identification of Memory Pressure

Subproblem 2 : A self-supervised approach to detect early signs of memory pressure

❑ *n* timestamps of observation, with time-gap of *k* timestamps, are our independent variables

❑ A balanced and normalized dataset for reduced bias

❑ Best results were achieved using 2 phases (under 85% memory usage vs. over 85%), essentially making it a binary classifier.

Phase 1: Early Identification of Memory Pressure

Subproblem 2 : A self-supervised approach to detect early signs of memory pressure

❑ A simple 5-layer architecture, with one dropout layer in-between to prevent overfitting.

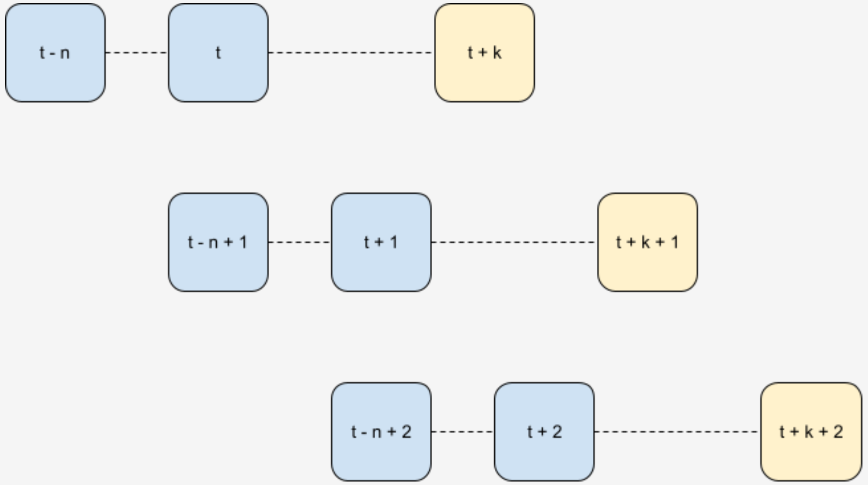
❑ Consistently reached ~86% accuracy. Other metrics are also showing promising results.

❑ Was reaching the minimum training loss within 50 epochs.

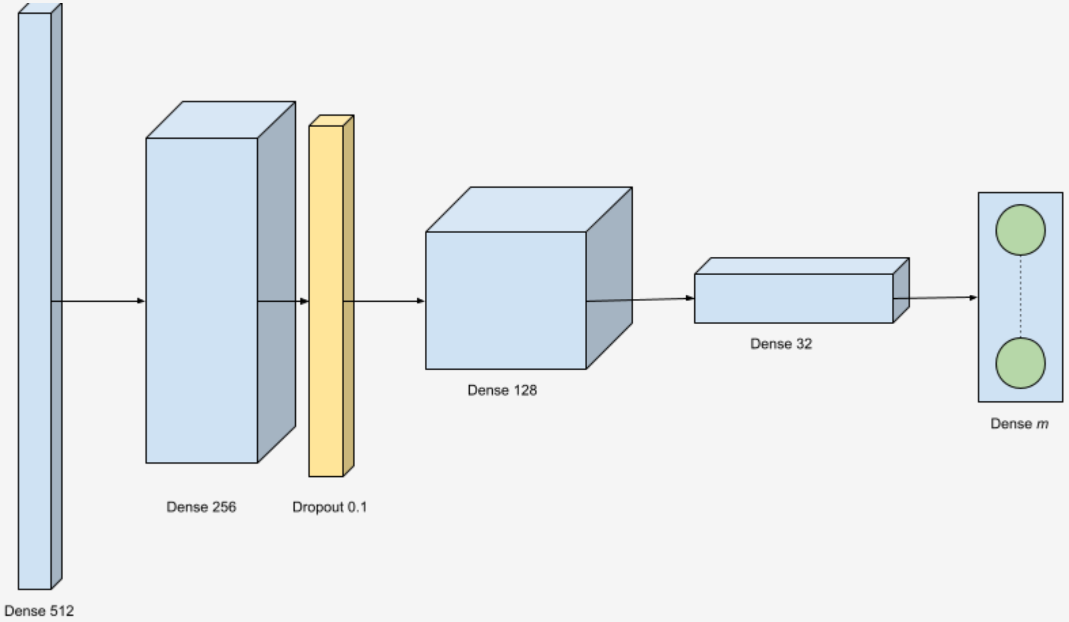
❑ An online approach has also been tested.

Phase 1: Early Identification of Memory Pressure

Subproblem 2 : A self-supervised approach to detect early signs of memory pressure



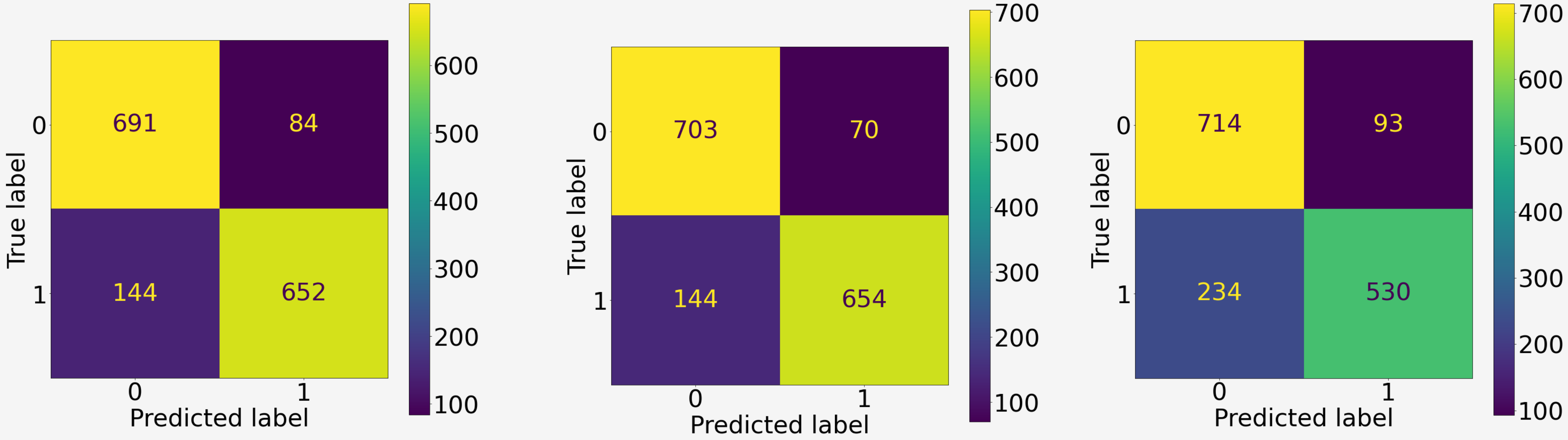
Data Representation



DNN Architecture

Phase 1: Early Identification of Memory Pressure

Subproblem 2 : A self-supervised approach to detect early signs of memory pressure



n = 3, k = 3

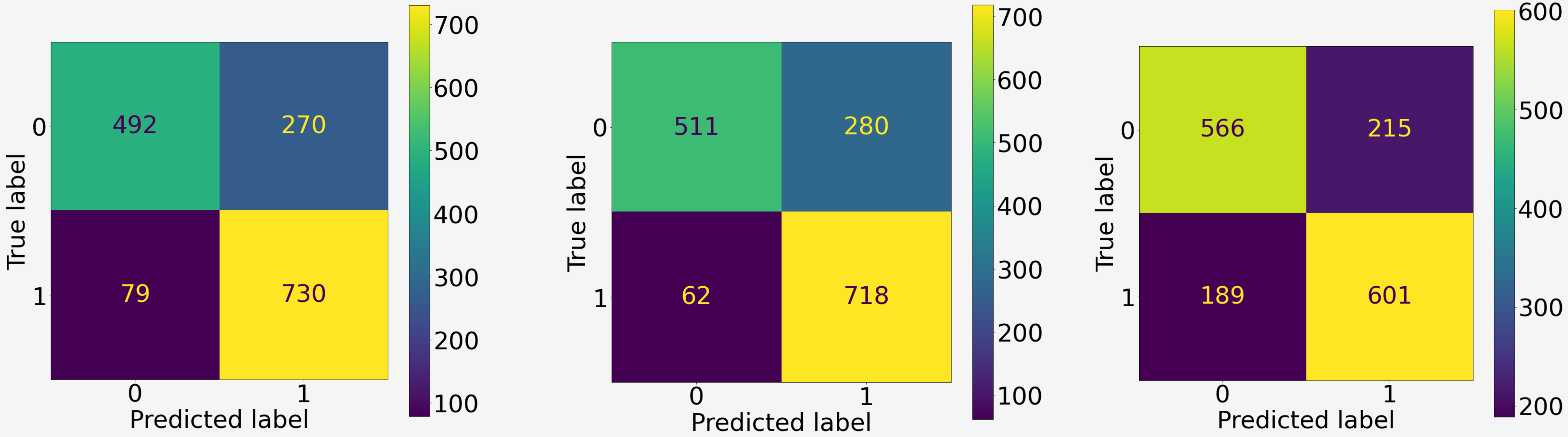
n = 4, k = 3

n = 4, k = 4

Test Data Performance

Phase 1: Early Identification of Memory Pressure

Subproblem 2 : A self-supervised approach to detect early signs of memory pressure



n = 3, k = 3, 78%

n = 4, k = 3, 85%

n = 4, k = 4, 60%

Test Data Performance with more tolerance to False-Positives

Phase 2: Taming Memory Pressure

- ❑ Start collecting process level information and make a memory usage profile for high memory occupying processes
- ❑ Bypass OOM killer to better identify culprit processes and kill
- ❑ Store information for debugging
- ❑ Challenges
  - ❑ System is already in stress, how much data do we want to collect
  - ❑ Offloading this task is not an option as there’s latency involved

Conclusion

❑ A difficult task indeed, but results so far are promising

❑ Overhead is still a big concern

❑ Production testing will give more insight

Questions?

1. Channabasappa, Smita Koralahalli. Performance Analysis and Control of Latency Under Memory Pressure in The Linux Kernel for Edge Computing. Diss. The University of North Carolina at Charlotte, 2019.
2. Zhuang, Zhenyun, et al. "Taming memory related performance pitfalls in linux Cgroups." 2017 International Conference on Computing, Networking and Communications (ICNC). IEEE, 2017.
3. Xin, Linlin, Hongjie Fan, and Zhiyi Ma. "An Optimization of Memory Usage Based on the Android Low Memory Management Mechanisms." International Conference on Mobile Computing, Applications, and Services. Springer, Cham, 2020.