UNC CHARLOTTE

The WILLIAM STATES LEE COLLEGE of ENGINEERING

Performance Analysis and Control of Latency under Memory Pressure in the Linux Kernel for Edge Computing

Smita Koralahalli Channabasappa

Department of Electrical and Computer Engineering
University of North Carolina at Charlotte

Advisor
Dr. Arun Ravindran

Committee
Dr. James Conrad
Dr. Hamed Tabkhi

SUMMARY

- Motivation
- Thesis Contribution
- Experimental Characterization of Latency Under Memory Pressure
- Proposed Research Design for the Latency Controller
- Advantages and Disadvantages of Proposed Research
- Related Work in Low Memory
- Conclusions and Future Work

Motivation

- Edge computing paradigm
  - Seeks to bring Cloud-like compute capabilities next to where the data is generated, to minimize the data communication latency.
- Edge applications such as autonomous driving, surveillance for accident and crime detection, and robotics are latency sensitive.
- Unlike cloud, edge applications are **resource constrained** making applications co-exist in same physical machines.
- **Memory** managed and virtualized by OS and not under applications control.
- High memory utilization maintaining latency constraints is a necessity in resource constrained platforms.

Thesis Contribution

- Linux kernel employs various policies targeting to improve system responsiveness and throughput.
- But high memory utilization and **latency constraints of individual applications** are not significantly taken care.
- This thesis provides **experimental evaluation** of the impact of different types of co-located memory intensive applications on the latency sensitive application.
- **Proposes a latency controller** under high memory pressure ensuring high memory utilization while not significantly violating latency constraints.
- YOLOv3, a Deep learning object detection application is chosen as latency critical as it is a part of many machine vision computing pipelines.

Resources employed for Study

- Dell laptop with Intel Core i7 CPU with 16GB memory.
- Nvidia GEFORCE GTX 1060 GPU.
- Ubuntu 16.04 LTS with Linux kernel version 4.20.
- PyTorch-YOLOv3 as a latency sensitive application.
- Two Microbenchmarks to simulate memory pressures.
- PSI - Pressure Stall Information upstreamed Linux Kernel 4.20.
- Virtual Memory Kernel parameters.
- BCC tools.
- cgroups.

Characterization of YOLOv3

YOLOv3

- ❑ Deep Learning based real-time object detector
- ❑ Part of many machine vision computing pipelines at the Edge

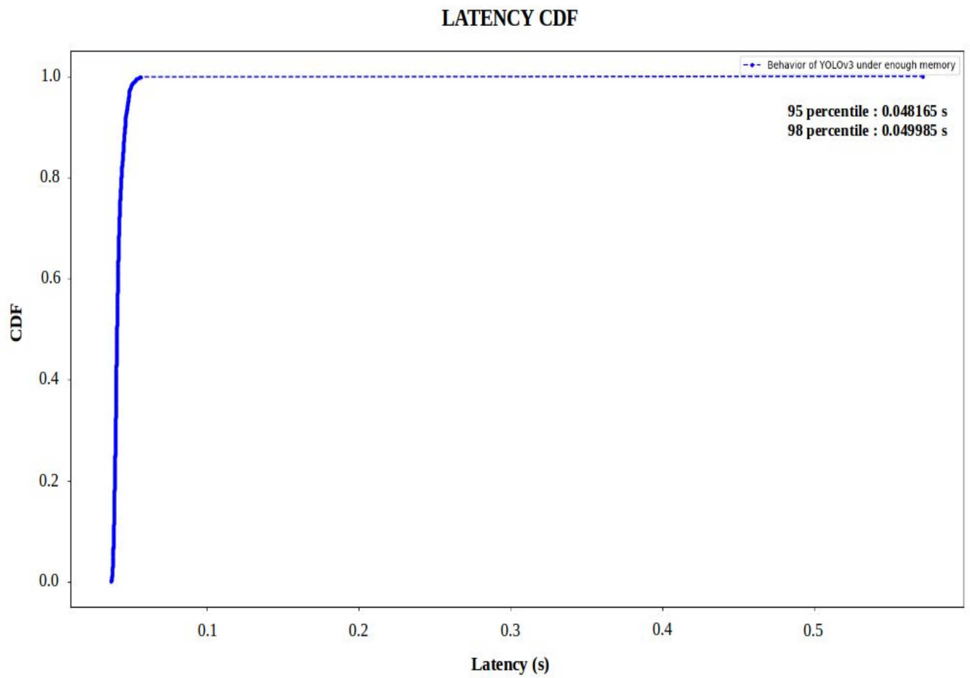

Fig 1: Latency CDF for YOLOv3 on a lightly loaded machine

- ❑ Consists of a series of 1000 images each with 100 KB to 200 KB in size.
- ❑ The inference latency for each object detection ranges from 0.03 - 0.05 seconds.
- ❑ 95th percentile is **0.04 seconds.**

Characterization of YOLOv3 (cont'd)

I/O Intensive application

- 1000x increase in page_ins made in the system when YOLOv3 started running.
- Consumes around 1.1GB for processing images with **0.98GB RSS for pretrained weights.**

Favors page_cache

- page_cache size increase from 1.5GB to 2.7GB when memory RSS of YOLOv3 increased.
- Miss rates in page_cache is on an average of 9% when YOLOv3 is being executed.

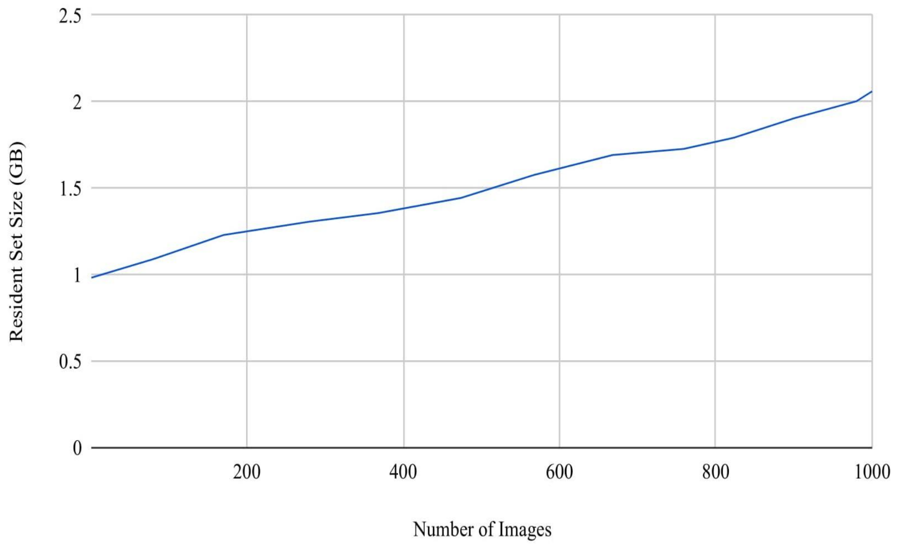

Fig 2: Memory Consumption of YOLOv3 (Resident Set Size) over number of Images

Experimental characterization of YOLOv3 under memory pressure

Impact of YOLOv3 inference latency under memory pressure

Memory Pressure can result either from

- ❑ Extensive consumption of Anonymous Pages in RAM.
- ❑ Extensive consumption of File backed Pages in Page_cache.

Experiment 1

- ❑ Running YOLOv3 and Anonymous Page Consumer Microbenchmark simultaneously

Experiment 2

- ❑ Running YOLOv3 and File backed Page Consumer Microbenchmark simultaneously
  - ❑ Examine the effects separately when page in File backed pages are accessed once and when pages in File backed pages accessed repeatedly

Experimental observation 1 : Effects on latency under the influence of Anonymous Page Memory Consumer Workload

- Latency at 95th percentile
  - 0.6 seconds, 20x increase

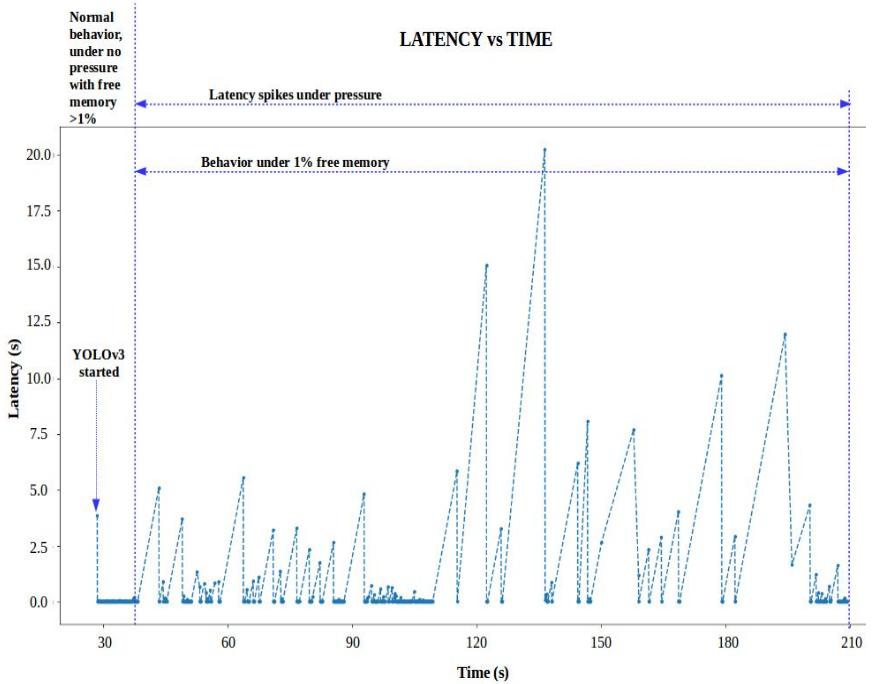

Fig 3: Time series plot of the inference latency suffered by YOLOv3 under the effect of Anonymous Page Consumer Workload

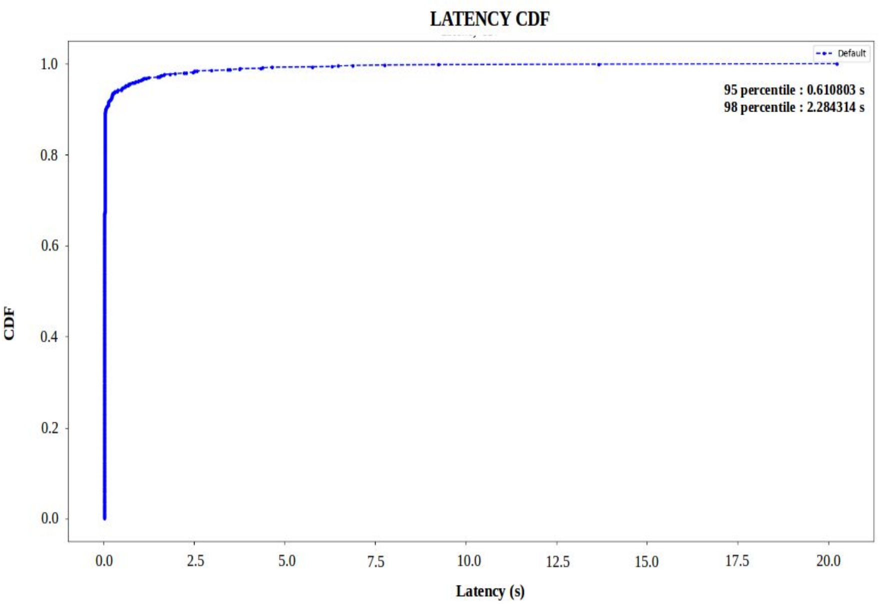

Fig 4: Latency CDF for YOLOv3 under memory pressure simulated by Anonymous page memory consumer microbenchmark

Other Significant Observations

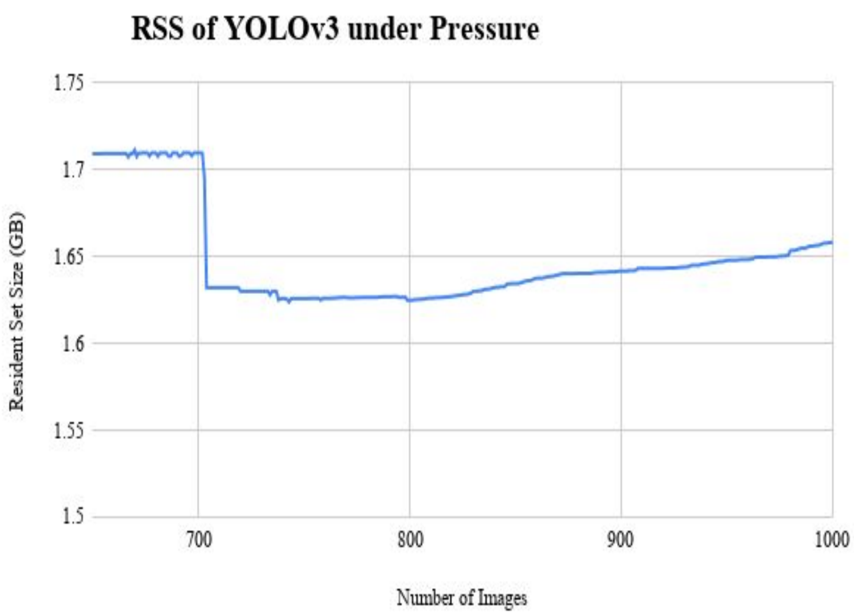

Fig 5: Effects on Resident Set Size (Anonymous+File backed) of YOLOv3 under the influence of Anonymous Page Consumer Microbenchmark

- Average miss rates in page_cache increased to 38% which is almost 3x as compared to running it under isolation.
- Page cache size also showed significant decrease.

Experimental observation 2 : Effects on latency under the influence of File backed Page Memory Consumer Workload when it is accessed repeatedly

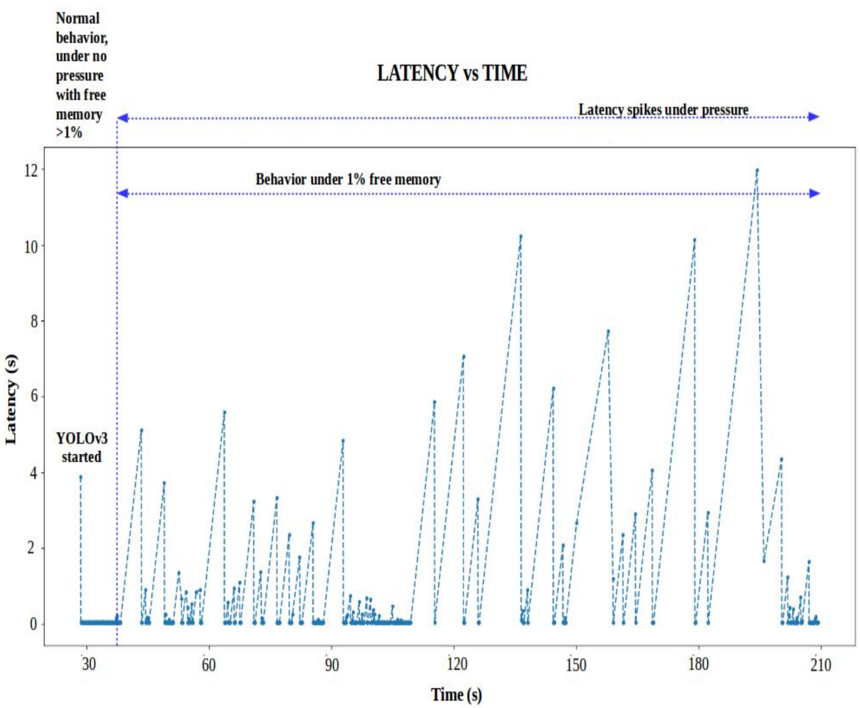

Fig 6: Time series plot of the inference latency suffered by YOLOv3 under the effect of File backed Page Consumer Workload

- Latency at 95th percentile
  - 0.32 seconds, 10x increase

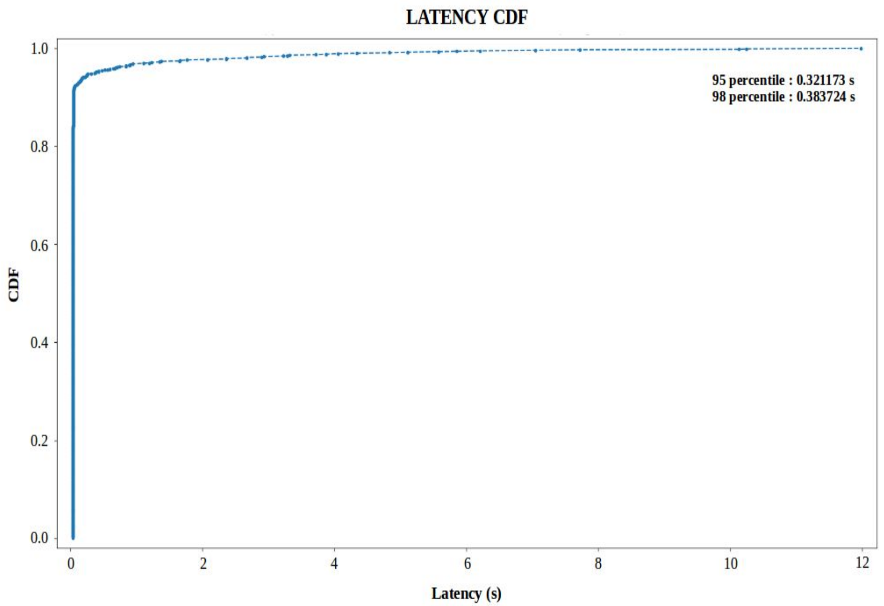

Fig 7: Latency CDF for YOLOv3 under memory pressure simulated by File backed page memory consumer microbenchmark

Other Significant Observations

When File backed pages are accessed once

- ❑ No significant pressure increase was observed.
- ❑ Decrease in Resident Set Size of Microbenchmark from 12.8 GB to 12.2 GB - page replacement policy chose inactive pages.

When File backed pages are accessed continuously

- ❑ Average miss rates in page_cache increased to 34.92% which is almost 3x as compared to running it under isolation.

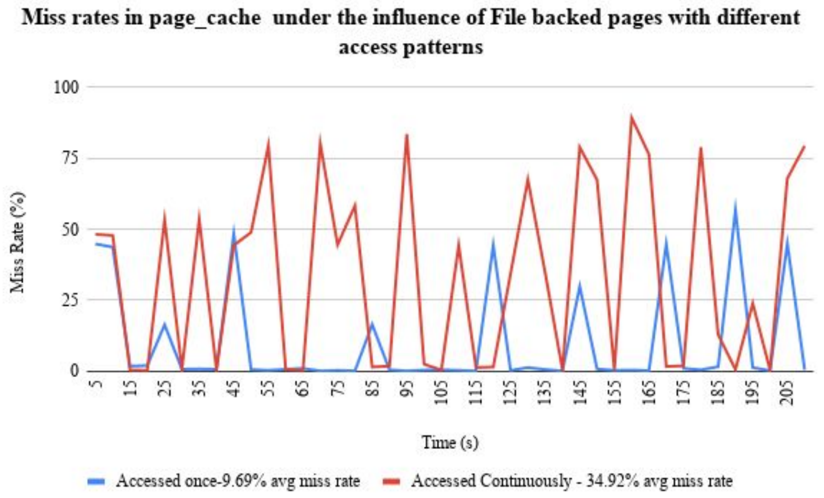

Fig 8: Miss rates in page_cache under the influence of file backed pages accessed once and accessed continuously

Proposed Research Design for the Latency Controller

- **Why research is needed?**
  - Adverse impact of high memory pressure on latency of an object-detection application (YOLOv3).

- **Two different control strategies are proposed to mitigate the latency violations allowing multiple applications to coexist on the same hardware at the Edge.**
  - *LATD* Controller to reduce latency under the effect of Anonymous Page Consumer Microbenchmark.

  - *cgroups* to reduce latency under the effect of File backed Page Consumer Microbenchmark.

Metrics used for implementation

PSI - Pressure Stall Information upstreamed kernel 4.20.

- **PSI *some***
  - Time in which at least one task is stalled on the resource.
- **PSI *full***
  - Time in which no task is making progress due to pressure.

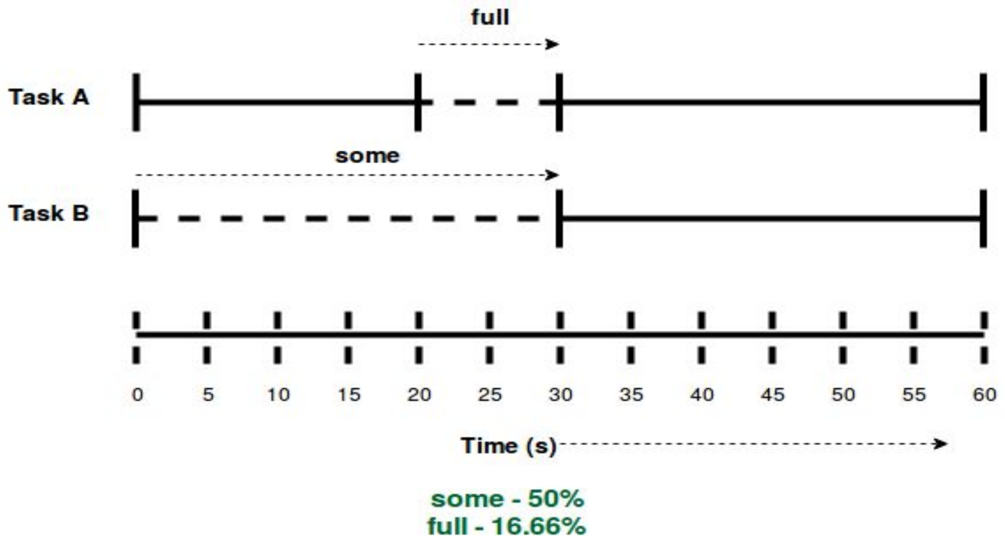

Fig 10: PSI *some* and *full* metrics

Metrics used for implementation (cont'd)

Virtual Memory kernel parameters

- Swappiness (Default - 60)
  - Controls size of page_cache. Tunable from 0 to 100.
- Dirty_background_ratio (Default 10)
  - Percentage of total available memory which when dirty, the system via flusher threads write dirty data to disk. Tunable from 0 to 100.
- Dirty_ratio (Default 20)
  - Percentage of total available memory which when dirty, the process generating the dirty pages, stops further I/O, and starts writing out dirty data.
- Overcommit_memory
  - Contains a flag for memory overcommitment.
    - 0 - kernel attempts to overcommit by estimating free memory.
    - 1 - Always overcommit.
    - 2 - the kernel uses a never overcommit policy.

LATD Controller for Anonymous Page Dominant Workload

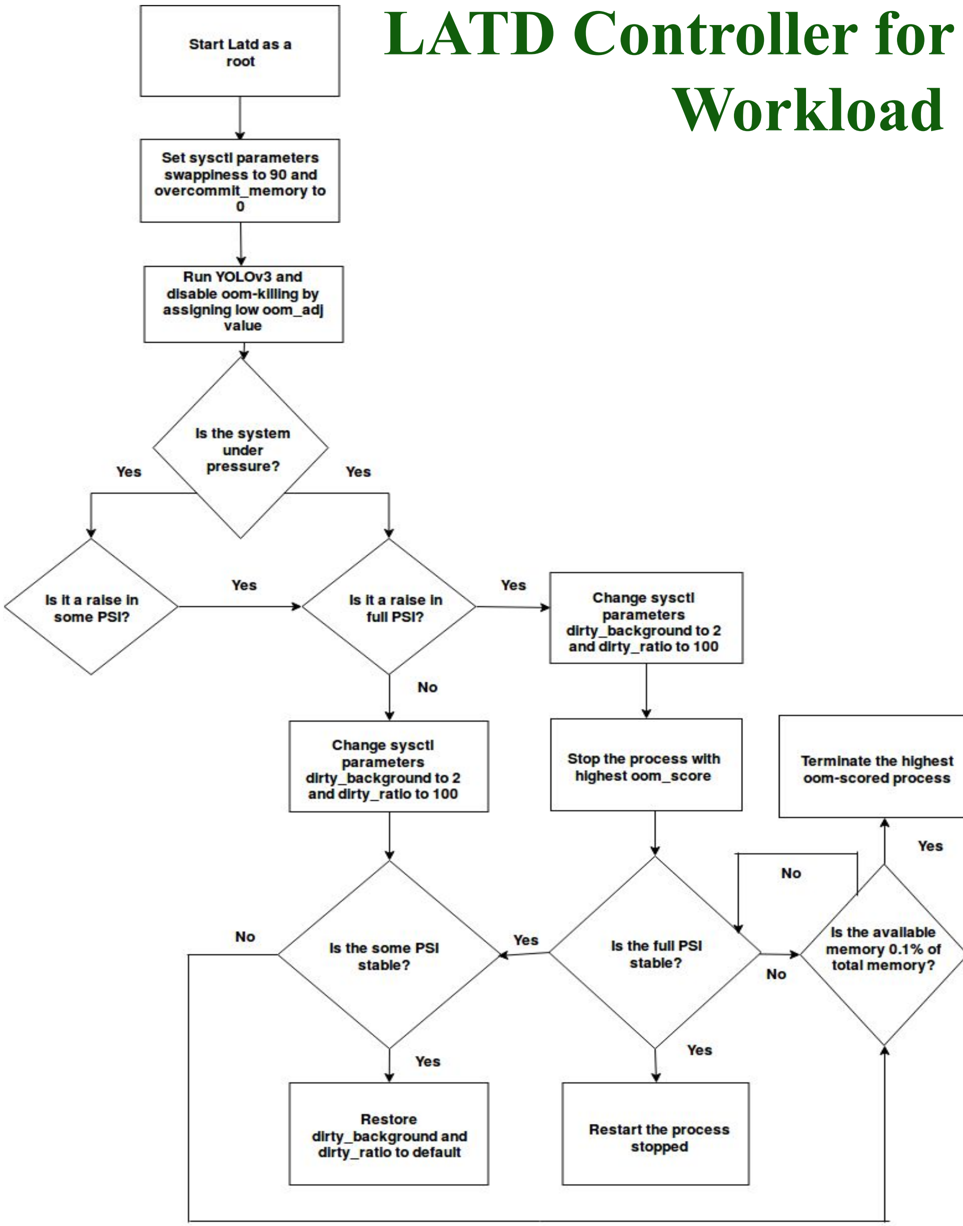

Fig 10: Flowchart of the implementation

- ❑ Controller is run as root to prioritize it over other applications.
- ❑ Disabled oom-killer for YOLOv3.
- ❑ Sysctl parameters are tuned for best efforts.
  - ❑ swappiness
  - ❑ dirty_background_ratio
  - ❑ dirty_ratio
  - ❑ overcommit_memory
- ❑ Memory pressure determined by PSI.
- ❑ Actions are enforced on bad process based on oom-score.

Results of LATD Controller

Advantages over non-controlled case

- **10x improvement** in latency.
- High memory utilization because of the corrective action taken

Disadvantages

- 2x degradation compared to lightly loaded system.
- Can be employed under the constraints that memory consumer benchmark is anonymous page dominant and latency sensitive application doesn't reuse dirty pages.

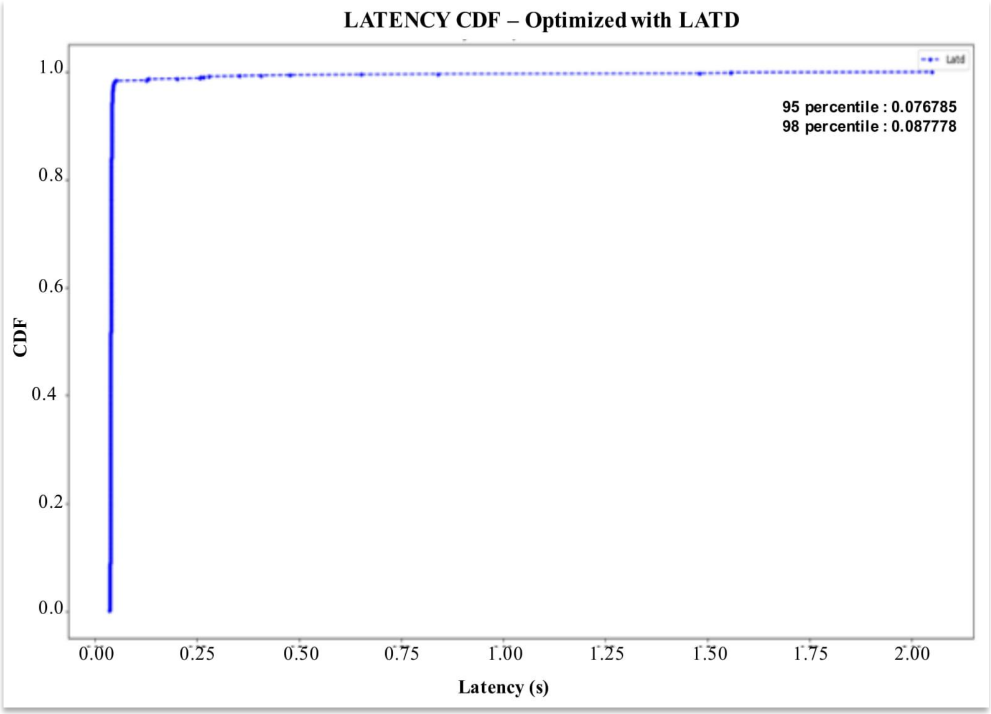

Fig 11: CDF Plot of Inference Latency of YOLOv3 after the effects of LATD Controller

- **Latency at 95th percentile**
  - 0.07 seconds, 10x improvement

Cgroups for File backed Page Memory Dominant Workload

- Latd Controller is inefficient for co-located workload that puts pressure on the page cache since we do not have control **over per-process use of the page cache** in the Linux kernel.

- Incorrectly tuning VM parameters might affect latency of latency critical applications.

- *cgroups* capability is explored which allows us to **specify memory resource limit** per process.

Cgroups for File backed Page Memory Dominant Workload

Steps employed

- Create two *cgroups* one for yolov3 and other for file backed page memory consumer microbenchmark.
- Set swappiness to 90 for YOLOv3 assigned cgroup and 0 to the microbenchmark.
- Set memory limits for cgroup where file backed page memory consumer microbenchmark is assigned.
- Start both applications
- Assign the applications to cgroups using their PID
- Stop highest oom-scored process when *some PSI* of the system increases and restart when it is stable.
- Terminate highest oom-scored process when available memory is 0.1% of total memory.

Effects on latency of yolov3 through cgroups

Advantages over non-controlled case

- Mitigate latency almost completely by limiting memory for the cgroup where microbenchmark is running.

Disadvantages

- No efficient memory utilization
  - Need to know consumption rates of latency critical and non latency critical applications ahead.

Possible Remedy

- Per *cgroup* pressure monitor. **(10 May 2019)**

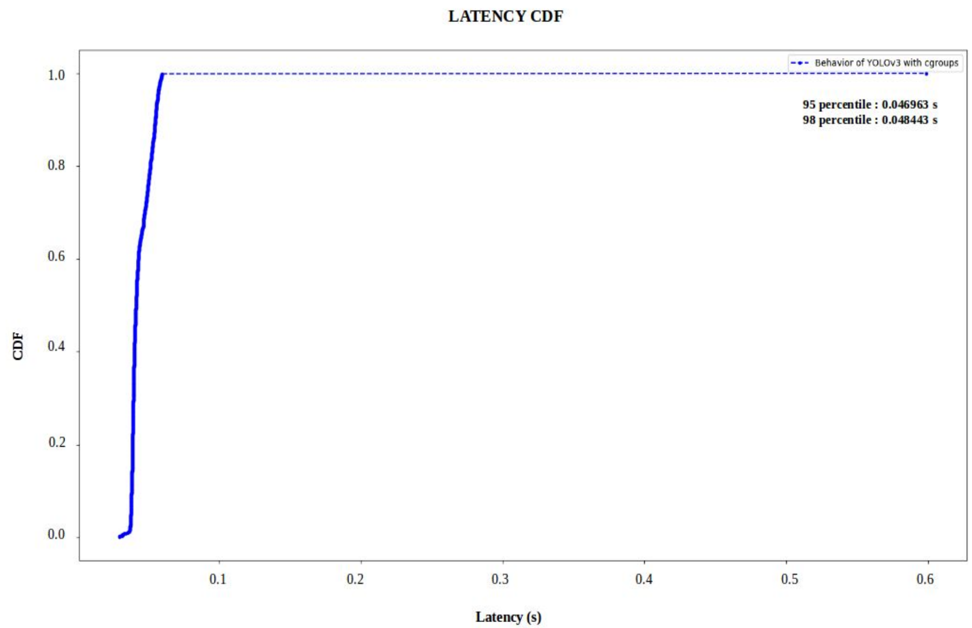

Fig 12: CDF Plot of Inference Latency of YOLOv3 placed under a cgroup

- **Latency at 95th percentile**
  - 0.04 seconds, similar to lightly loaded case

Possible direction of implementation

- The first 5 steps are followed as discussed before until the PID of the processes are assigned to particular cgroups.

- If *some* PSI of the file backed microbenchmark cgroup increases then spin up another cgroup with higher memory limits.
  - Assign the process affected by rise in *some* PSI to new cgroup
  - Destroy unused *cgroup* by after moving the process to release memory

- Stop highest oom-scored process when *some PSI* of the system increases and restart when it is stable.

- Terminate highest oom-scored process when available memory is 0.1% of total memory.

Applications

***LATD* Controller**

- If latency critical application is page cache page dominant which most real time applications are.
- If the background applications are more anonymous page dominant.
- If latency critical application has no good write hits indicating less re-usage of dirty pages.

cgroups

- If majority of the applications are file backed page dominant with frequent re-access of used pages.

Related Work

Oomd by Facebook - Leverages PSI and cgroupv2 to monitor a system and kill offending process in userspace.

Earlyoom - OOM preventer in user space by sending SIGTERM signal available as repository.

The above approaches

- ❑ Overcome oom livelocks and system freeze.
- ❑ Doesn’t overcome the latency suffered by individual application under pressure.

MISE: Providing performance predictability and improving fairness in shared main memory systems [IEEE 2013] - estimates slowdown using request service rates and assigning priorities.

Heracles: Improving resource efficiency at scale [ACM 2015] - isolation mechanism to enable a latency-critical workload to be colocated with batch jobs

Conclusions and Future Work

Conclusions

- Achieves practical deployment of latency sensitive applications along with memory intensive background applications on the same physical machine at the Edge while efficiently utilizing the memory.

- Reduces the impact on latency of latency critical applications due to memory intensive applications running alongside.

- Corrective actions taken, depends on the type of memory consumed by the background application - anonymous or file backed.

Future Work

- Comprehensive evaluation of our proposed latency controller with realistic background benchmarks.
- Per *cgroup* resource process monitoring based on newly available enhancement allowing containers to be resized dynamically based on the memory availability.

Thank you