Qualcomm

Enhancing the PSI framework in Linux Kernel for predictive and accurate workload analysis

Introducing AI in Linux Kernel

Pintu Kumar (Agarwal)

Engineer Staff, Qualcomm India Private Limited,

Bangalore

@qualcomm

Agenda

- Scope
- Introduction
- PSI Internals
- Block Diagram & Flow Chart
- Experimentation Results
- Summary and Conclusion
- Reference

Scope

- Limited to Linux Kernel 5.15 or below.
- Latest changes in PSI are not covered here.
- Most results shown here are derived from qemu, arm, systemd, ext4.
- Stress-ng is used to simulate heavy load on the system.
- Some test utilities were used to trigger use cases.
- System configuration:
  - RAM: 512MB
  - CPU: 4
  - Storage: 4GB

Introduction

- PSI : Pressure Stall Information related to CPU, Memory, IO usage.
- Available from Linux Kernel 4.20 onwards.
- PSI monitors these resource contention and provides real-time insights into system performance bottlenecks.
- It just gives the overall average load value in the system during certain intervals.
- This paper covers advancements in the PSI framework within the Linux kernel to enhance predictive workload analysis.
- Aim is to provide more granular information of system congestion at each task level.

How PSI Works

- PSI works by collecting average load information for cpu, memory and io.
- It gives the overall average load value in the system in the last 10 seconds,

  1 min (60s) and 5 min (300s) duration of resource usage.

/ # cat /proc/pressure/**cpu** **some** avg10=0.08 avg60=0.02 avg300=0.00 **total**=39360 **full** avg10=0.00 avg60=0.00 avg300=0.00 total=0	/ # cat /proc/pressure/**memory** **some** avg10=0.01 avg60=0.02 avg300=0.00 total=52645 **full** avg10=0.00 avg60=0.00 avg300=0.00 total=10155	/ # cat /proc/pressure/**io** **some** avg10=0.00 avg60=0.02 avg300=0.00 total=29723 **full** avg10=0.00 avg60=0.00 avg300=0.00 total=0

- **Some** indicates: some tasks in the system are delayed in these intervals.
- **Full** indicates: almost all tasks are delayed due to lack of resource.
- The value indicates the percentage of time the task got delayed.
- The load values are calculated based on CALC_LOAD formula in Linux Kernel.
- The total indicates the time in microseconds.

PSI Load Calculation Formula

LOAD_FREQ (F) = 1sec

Precision bits = 11

Value of e = 2.71828182845

Load Interval Formula

$$EXP\_Xs = 2^{11} / e^{(\frac{F}{X})}$$

Load Interval	EXP Value (for 1sec interval)	EXP Value (for 2sec interval)
1 sec	$EXP\_1s = 2^{11} / e^{(\frac{1}{1})}$ = 2048 / 2.7182 = 753	$EXP\_1s = 2^{11} / e^{(\frac{2}{1})}$ = 2048 / 2.7182 = 277
5 sec	$EXP\_5s = 2^{11} / e^{(\frac{1}{5})}$ = 2048 / 1.2214 = 1677	$EXP\_5s = 2^{11} / e^{(\frac{2}{5})}$ = 2048 / 1.2214 = 1373
10 sec	$EXP\_10s = 2^{11} / e^{(\frac{1}{10})}$ = 2048 / 1.1051 = 1853	$EXP\_10s = 2^{11} / e^{(\frac{2}{10})}$ = 2048 / 1.1051 = 1677
60 sec	$EXP\_60s = 2^{11} / e^{(\frac{1}{60})}$ = 2048 / 1.1680 = 2014	$EXP\_60s = 2^{11} / e^{(\frac{2}{60})}$ = 2048 / 1.1680 = 1981
300 sec	$EXP\_300s = 2^{11} / e^{(\frac{1}{300})}$ = 2048 / 1.0033 = 2041	$EXP\_300s = 2^{11} / e^{(\frac{2}{300})}$ = 2048 / 1.0033 = 2034

These values are used inside PSI to calculate the load average for each resource

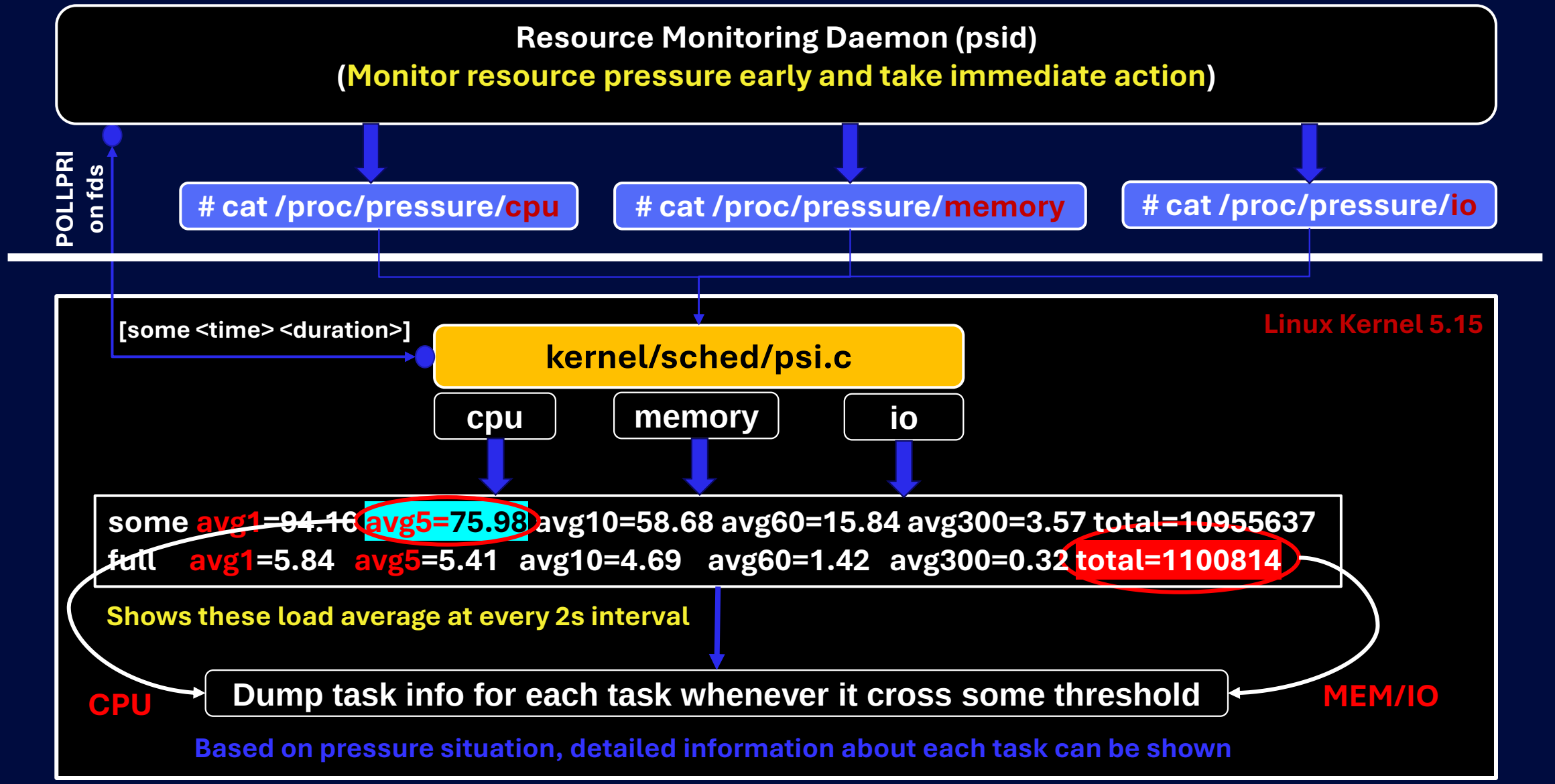
```
CALC_LOAD(avenrun[0], EXP_1, active_tasks)
CALC_LOAD(avenrun[0], EXP_5, active_tasks)
CALC_LOAD(avenrun[0], EXP_10, active_tasks)
```

Reference: https://www.linuxjournal.com/article/9001

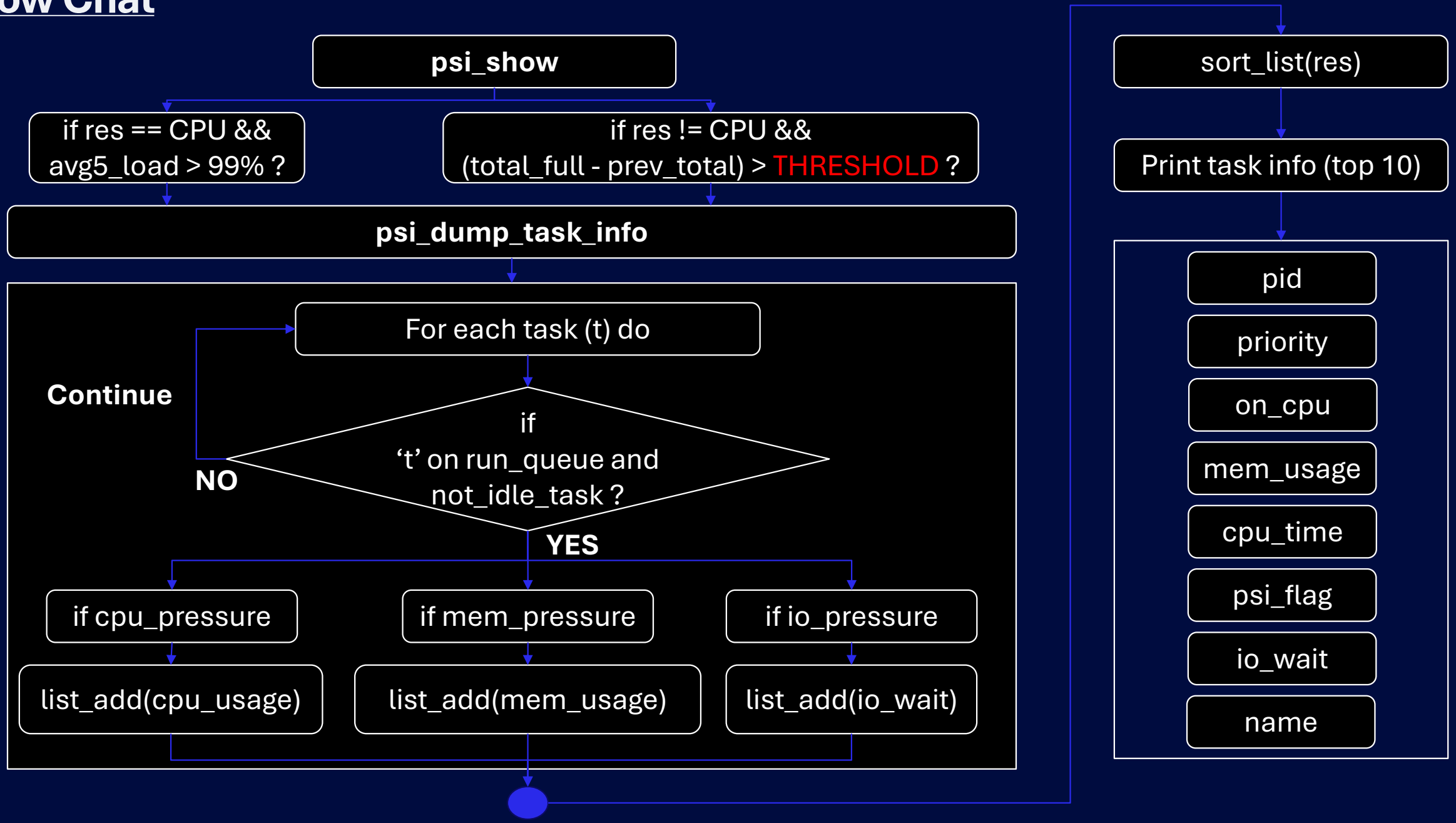
PSI Changes

- Added 1sec and 5sec average load data for quick results.
- Change load frequency from 2sec to 1sec (for experiment data only).
- Introduce threshold criteria based on current and previous load status.
- Introduce algorithm to track each non-idle task on run queue and present more granular information about each task usage.
- Introduce new CONFIG_PSI_DUMP_TASK_INFO to enable these changes.
- Developed system-monitor daemon to trace these information in user-space, including poll events.

Block Diagram



Flow Chat


psi_show
if res == CPU && avg5_load > 99% ?
if res != CPU && (total_full - prev_total) > THRESHOLD ?
psi_dump_task_info
For each task (t) do
Continue
if 't' on run_queue and not_idle_task ?
NO
YES
if cpu_pressure
if mem_pressure
if io_pressure
list_add(cpu_usage)
list_add(mem_usage)
list_add(io_wait)
sort_list(res)
Print task info (top 10)
pid
priority
on_cpu
mem_usage
cpu_time
psi_flag
io_wait
name

Experimentation Results

Results – CPU Test 1

/home/pintu # **./stress-ng** -a 0 --cpu 5 --vm 5 --iomix 2 --iomix-bytes 100M –cpu-load 100 --daemon 50 --matrix 10 --pathological --aggressive --timeout 1m &

/home/pintu # **cat /proc/pressure/cpu (Every 1 sec)**

PSI_FLAGS

NR_ONCPU	NR_RUNNING	MEMSTALL	IOWAIT	DECIMAL
bit 3	bit 2	bit 1	bitt 0	
1	1	0	0	12
0	1	0	0	4

```
some avg1=100.00 avg5=99.32 avg10=92.14 avg60=34.67 avg300=8.41 total=26940476
   full avg1=    0.00 avg5=    0.00 avg10=  0.00 avg60=   0.00 avg300=0.00 total=0

PID=108  PRIO=120  ON_CPU=3  MEM(kB)=  2280  CPUTIME(ms)=1560  PSIFLAG=4  COMM=stress-ng
PID=109  PRIO=120  ON_CPU=3  MEM(kB)=  2280  CPUTIME(ms)=1540  PSIFLAG=4  COMM=stress-ng
[...]
PID=128  PRIO=120  ON_CPU=1  MEM(kB)=21468  CPUTIME(ms)=1540  PSIFLAG=4  COMM=stress-ng
PID=130  PRIO=120  ON_CPU=0  MEM(kB)=20940  CPUTIME(ms)=1610  PSIFLAG=4  COMM=stress-ng
PID=131  PRIO=120  ON_CPU=2  MEM(kB)=21336  CPUTIME(ms)=1630  PSIFLAG=4  COMM=stress-ng
[...]
PID=136  PRIO=120  ON_CPU=2  MEM(kB)=21204  CPUTIME(ms)=1650  PSIFLAG=4  COMM=stress-ng
[...]
PID=332  PRIO=120  ON_CPU=2  MEM(kB)=  1048  CPUTIME(ms)=1470  PSIFLAG=4  COMM=stress-ng
PID=381  PRIO=120  ON_CPU=3  MEM(kB)=     4  CPUTIME(ms)=   10  PSIFLAG=12 COMM=cat
....
Total tasks on run queue = 71
```

Results – Memory Test 1

/home/pintu # **./stress-ng** -a 0 --cpu 5 --vm 2 --io 5 --hdd 2 --cpu-load 90 --daemon 5 --matrix 5 --pathological --aggressive --timeout 2m &
/home/pintu # **cat /proc/pressure/memory (Every 1 sec)**

PSI_FLAGS

MEMSTALL_RUNNING	ONCPU	RUNNING	MEMSTALL	IOWAIT	DECIMAL
bit4	bit 3	bit 2	bit 1	bitt 0	
1	0	1	1	0	22
0	1	1	0	0	12
0	0	1	0	0	4

```
        total     used     free     shared    buff/cache    available
Mem:    496       458      7        2         30            19

some    avg1=14.03    avg5=6.70    avg10=4.82    avg60=3.17    avg300=1.01    total=3612844
full    avg1= 1.74    avg5=0.81    avg10=0.56    avg60=0.41    avg300=0.13    total=773131
total_full = 773131 ; prev_total_full = 616157  (~156 ms)

PID=1      PRIO=120  ON_CPU=1  MEM(kB)=3472     CPUTIME(ms)=28920  PSIFLAG=4   COMM=systemd
PID=47     PRIO=120  ON_CPU=3  MEM(kB)=0        CPUTIME(ms)=2220   PSIFLAG=22  COMM=kswapd0
PID=175    PRIO=120  ON_CPU=3  MEM(kB)=4        CPUTIME(ms)=14410  PSIFLAG=4   COMM=infinite-loop.o
PID=180    PRIO=120  ON_CPU=1  MEM(kB)=103288   CPUTIME(ms)=7530   PSIFLAG=4   COMM=thread-alloc.ou
PID=177    PRIO=120  ON_CPU=3  MEM(kB)=2860     CPUTIME(ms)=8050   PSIFLAG=4   COMM=stress-ng
PID=197    PRIO=120  ON_CPU=3  MEM(kB)=109980   CPUTIME(ms)=13960  PSIFLAG=4   COMM=stress-ng
PID=200    PRIO=120  ON_CPU=2  MEM(kB)=118836   CPUTIME(ms)=14040  PSIFLAG=4   COMM=stress-ng
PID=213    PRIO=120  ON_CPU=0  MEM(kB)=720      CPUTIME(ms)=15290  PSIFLAG=4   COMM=stress-ng
PID=214    PRIO=120  ON_CPU=1  MEM(kB)=720      CPUTIME(ms)=14670  PSIFLAG=12  COMM=stress-ng
PID=10652  PRIO=120  ON_CPU=2  MEM(kB)=364      CPUTIME(ms)=50     PSIFLAG=12  COMM=cat
.....
Total tasks on run queue = 20
```

This indicates, task stuck on run queue due to memory stall, thus CPU consumptions are also high.

Results – IO Test 1

/home/pintu # **./stress-ng** -a 0 --cpu 5 --vm 6 --io 10 --hdd 4 --cpu-load 100 --daemon 50 --matrix 10 --pathological --aggressive --timeout 1m &
/home/pintu # **cat /proc/pressure/io**

PSI_FLAGS

MEMSTALL_RUNNING	ONCPU	RUNNING	MEMSTALL	IOWAIT	DECIMAL
bit4	bit 3	bit 2	bit 1	bitt 0	
1	1	1	1	0	30
1	0	1	1	0	22
0	1	1	0	0	12
0	0	1	0	0	4

```
some avg1=80.25 avg5=60.12 avg10=57.79 avg60=39.85 avg300=16.74 total=111871130
full    avg1=  0.86 avg5=  0.87 avg10=   3.00 avg60=  3.63 avg300=   1.16 total=8476803
total_full = 8476803 ; prev_total_full = 8453359

PID=     47  PRIO=120  ON_CPU=1  MEM(kB)=        0  CPUTIME(ms)=  7470  PSIFLAG= 30  COMM=kswapd0
PID=19893  PRIO=120  ON_CPU=0  MEM(kB)=     2656  CPUTIME(ms)=  3510  PSIFLAG=  4  COMM=stress-ng
PID=19894  PRIO=120  ON_CPU=0  MEM(kB)=     2160  CPUTIME(ms)=  8340  PSIFLAG=  4  COMM=stress-ng
PID=19895  PRIO=120  ON_CPU=0  MEM(kB)=     2212  CPUTIME(ms)=  8220  PSIFLAG= 12  COMM=stress-ng
[...]
PID=19914  PRIO=120  ON_CPU=2  MEM(kB)=    97712  CPUTIME(ms)=12560  PSIFLAG=  4  COMM=stress-ng
PID=19915  PRIO=120  ON_CPU=2  MEM(kB)=      288  CPUTIME(ms)=  8690  PSIFLAG=  4  COMM=stress-ng
[...]
PID=19952  PRIO=120  ON_CPU=1  MEM(kB)=101392  CPUTIME(ms)=  5460  PSIFLAG=22  COMM=stress-ng
PID=19962  PRIO=120  ON_CPU=2  MEM(kB)=      352  CPUTIME(ms)=    60  PSIFLAG= 12  COMM=cat

Total tasks on run queue = 24
```

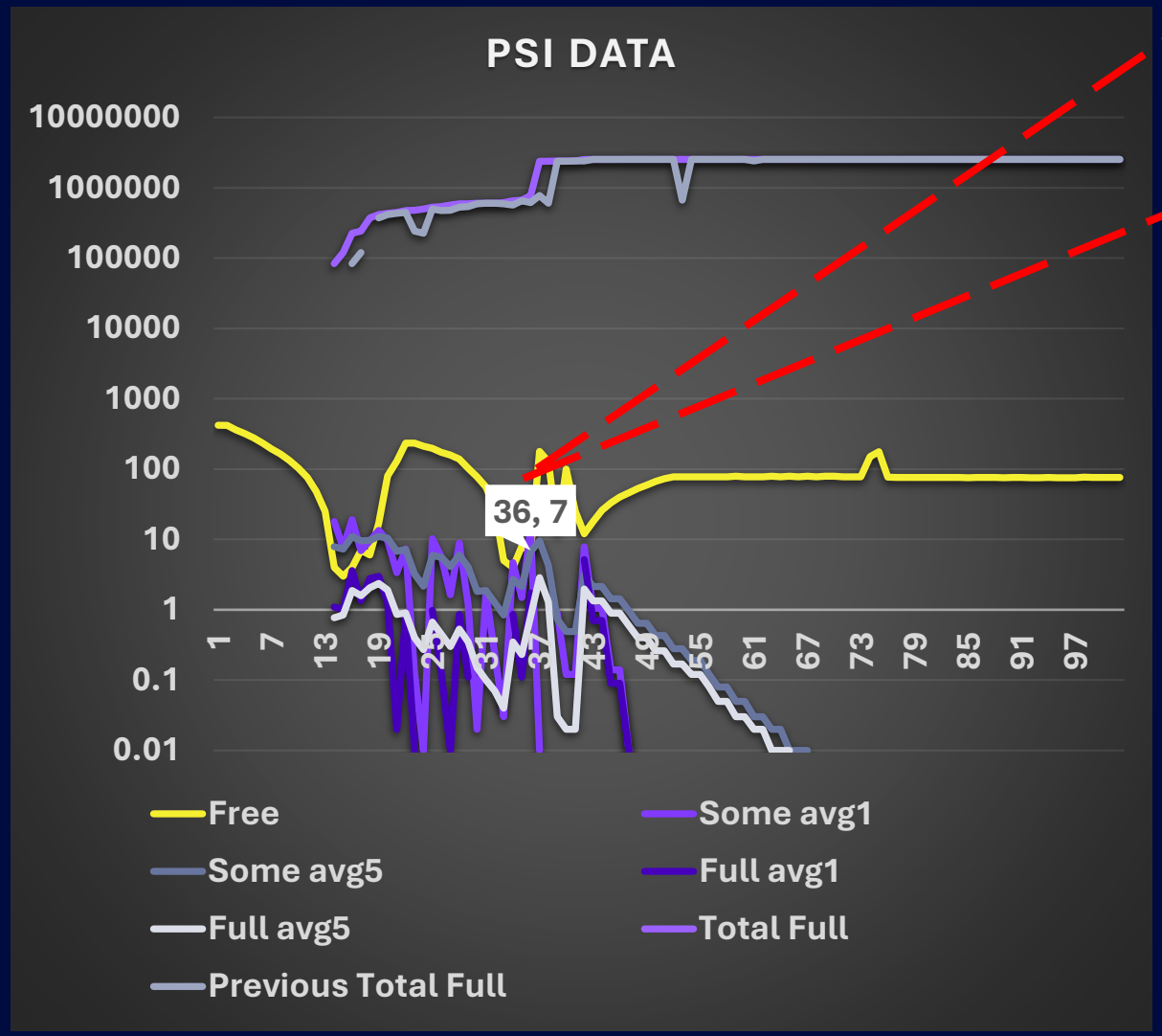
Interesting Results

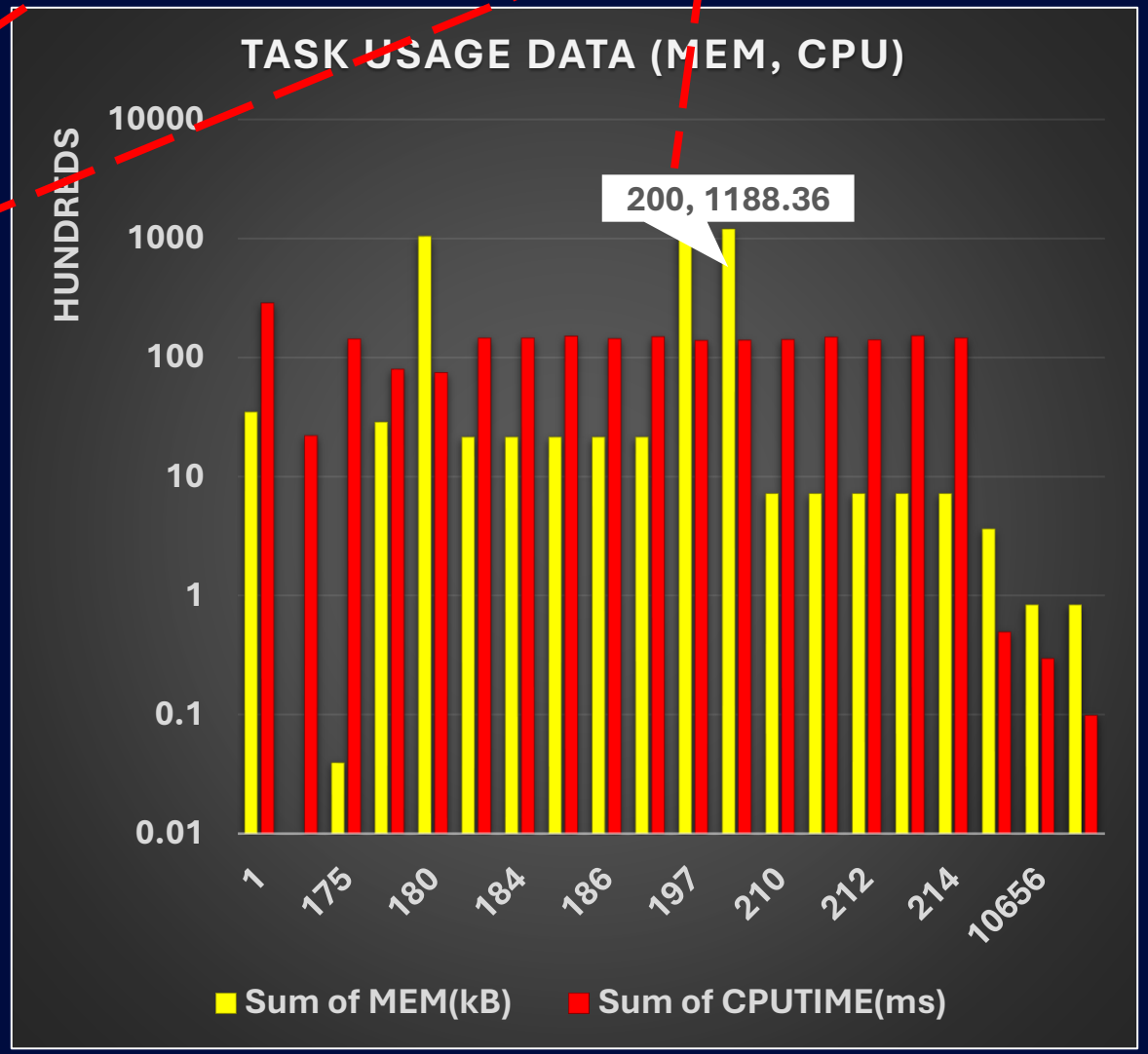
```
              total        used        free      shared  buff/cache   available
Mem:            496         417           4           7          74          60

PID=19952  PRIO=120  ON_CPU=1  MEM(kB)=101392  CPUTIME(ms)=5460  PSIFLAG=22  COMM=stress-ng
PID=19962  PRIO=120  ON_CPU=2  MEM(kB)=     352  CPUTIME(ms)=   60  PSIFLAG= 12  COMM=cat

Out of memory: Killed process 19952 (stress-ng) total-vm:145824kB, anon-rss:130008kB,

              total        used        free      shared  buff/cache   available
Mem:            496         354         120           7          21         124

some avg1=58.82 avg5=72.67 avg10=74.29 avg60=51.21 avg300=21.10 total=128976436
full avg1=35.70 avg5=40.94 avg10=36.38 avg60=13.74 avg300=3.67 total=16762002
total_full = 16762002 ; prev_total_full = 8476803  (~8 secs)

PID=19965  PRIO=120  ON_CPU=2  MEM(kB)=332  CPUTIME(ms)=20  PSIFLAG=12  COMM=cat
```

System was low on memory, PSI detects the hogging process, OOM occurred and killed it.
CPU time of another process got reduced.

Memory Pressure(View)

This is where the OOM actually occurred





Summary / Conclusion

- We have seen how a simple enhancement in the PSI module can be helpful in quickly identify the pressure situation in a system and finding the right culprit on the spot itself.
- Using resource-monitor service these data can be automatically monitored in the background and reported to users based on the certain threshold.
- Further tuning might be needed for each system depending on workload.
- Tracking workload at user-space may not be feasible always during heavy pressure condition and user-space process may not be scheduled for longer time.
- Enhancing PSI is surely going to be beneficial for small embedded devices.
- As a further enhancement we can also think of introducing cat /proc/pressure/**network** to monitor network/modem load.
- The key benefits are :- detecting system congestion early, avoid system hang or crash due to heavy pressure, quick debugging of critical issues, improve product reliability and quality in the long run, reduces postmortem analysis and efforts.
- Leaving here with the thoughts of introducing in-kernel advanced artificial intelligence algorithm to predict system congestion early and take appropriate action.

References

- https://docs.kernel.org/accounting/psi.html
- https://facebookmicrosites.github.io/psi/docs/overview.html
- https://unixism.net/2019/08/linux-pressure-stall-information-psi-by-example/
- https://man7.org/linux/man-pages/man8/systemd-oomd.service.8.html
- https://www.dbi-services.com/blog/pressure-stall-information-on-autonomous-linux/
- https://github.com/holmanb/psimon

For further queries, please get in touch on LinkedIn:
https://www.linkedin.com/in/pintu-agarwal-b73a31b/

Thank you

Follow us on:

For more information, visit us at qualcomm.com & qualcomm.com/blog