cgroup2 Resource Control strategies w/ resctl-demo

Tejun Heo

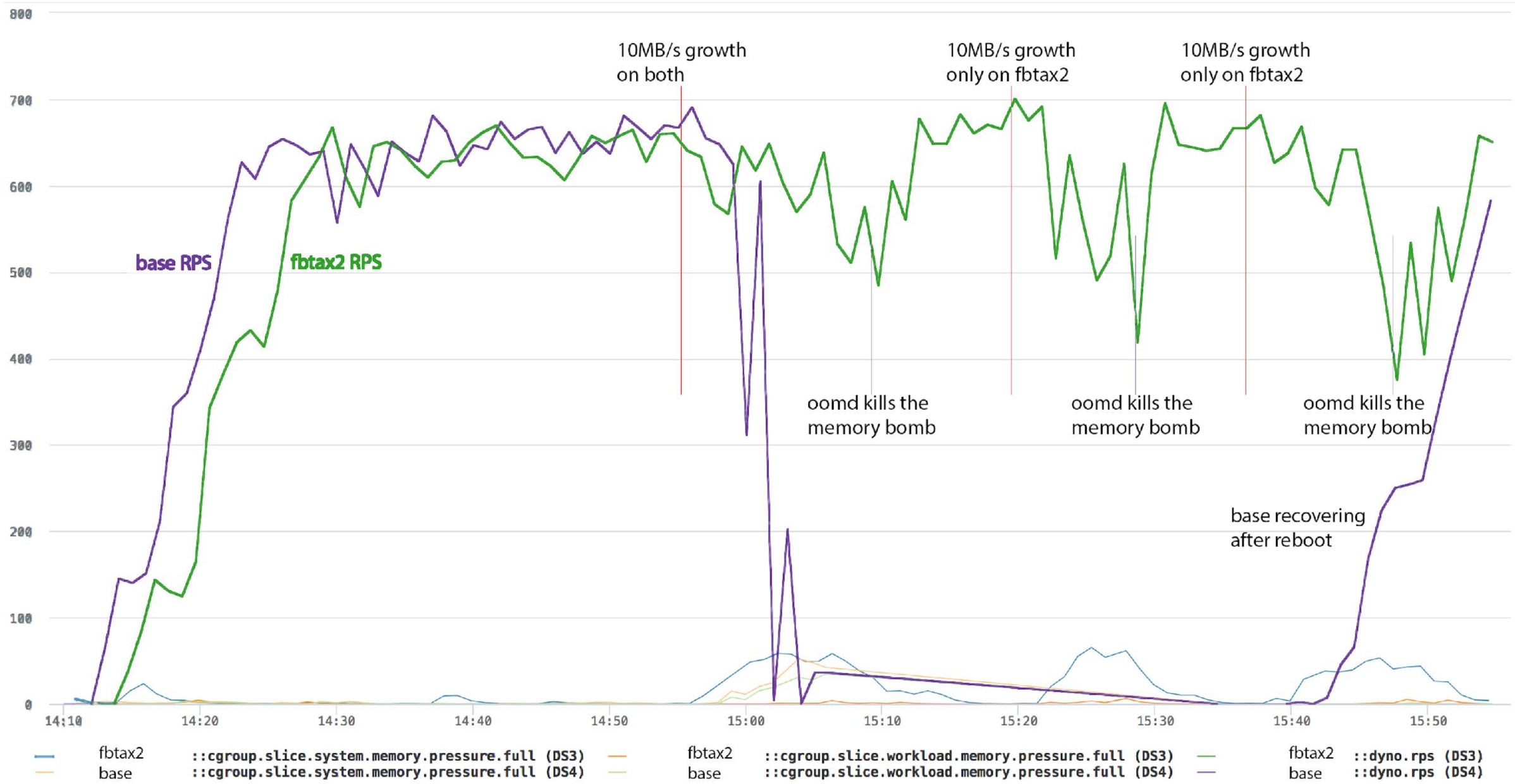
Software Engineer

The Goal of resource control

Work-conserving full-OS resource isolation

What does that mean?

- Work-conserving:
  - Don’t keep machine idle if there’s work to do
- Full-OS:
  - Transparent
  - Keep doing what you’ve been doing and overlay isolation
  - No need for direct IO, hard-allocate mem, separate FS

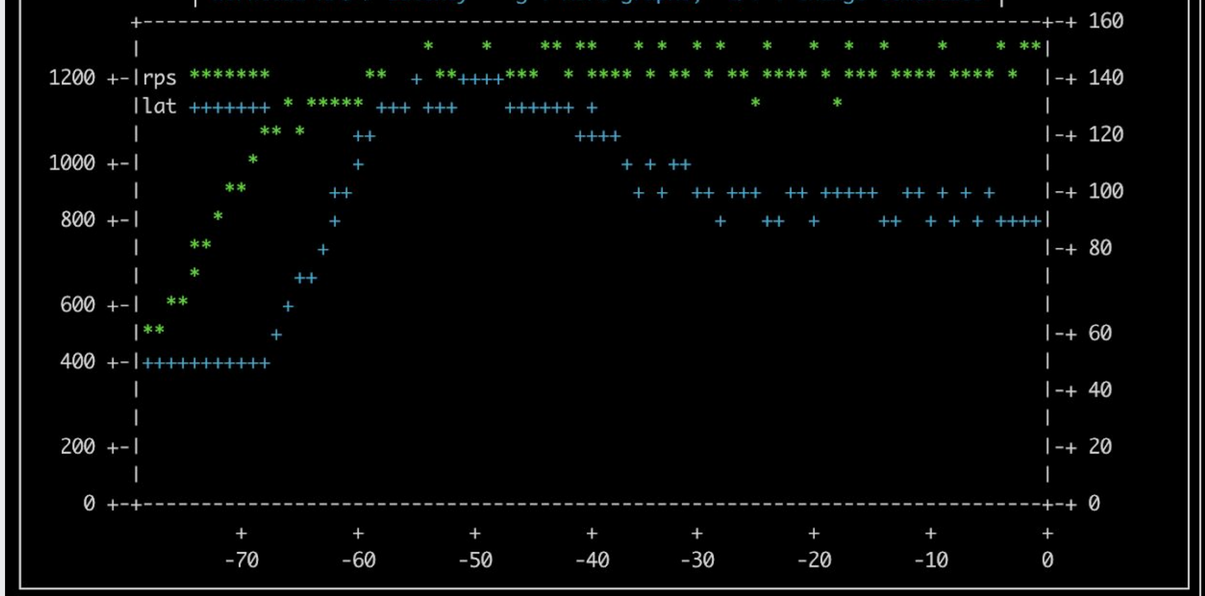

800
700
600
500
400
300
200
100
0
14:10
14:20
14:30
14:40
14:50
15:00
15:10
15:20
15:30
15:40
15:50
10MB/s growth
on both
10MB/s growth
only on fbtax2
10MB/s growth
only on fbtax2
base RPS
fbtax2 RPS
oomd kills the
memory bomb
oomd kills the
memory bomb
oomd kills the
memory bomb
base recovering
after reboot
fbtax2 ::cgroup.slice.system.memory.pressure.full (DS3)
base ::cgroup.slice.system.memory.pressure.full (DS4)
fbtax2 ::cgroup.slice.workload.memory.pressure.full (DS3)
base ::cgroup.slice.workload.memory.pressure.full (DS4)
fbtax2 ::dyno.rps (DS3)
base ::dyno.rps (DS4)

Facebook Resource Control Demo v1.0.0 - 'q': quit

```
[  Running  ] 2021-01-22 03:13:48 PM
[  config   ] satisfied: 18  missed:  0
[   oomd    ] workload: +pressure -senpai  system: +pressure -senpai
[ sideload  ] jobs:  0/ 0  failed:  0  cfg_warn:  0  +overload -crit
[  sysload  ] jobs:  0/ 0  failed:  0
[ workload  ] load: 99.0%  lat: 100ms  cpu: 91.5%  mem: 28.3G  io: 16.5M
```

	cpu%	mem	swap	rbps	wbps	cpuP%	memP%	ioP%
workload	92.3	28.7G	856K	15.0M	3.9K	84.2	-	0.0
sideload	-	-	-	-	-	-	-	-
hostcritical	7.4	647M	-	-	-	31.7	-	0.0
system	0.0	25.8M	1.5M	-	-	0.1	-	-
user	0.0	577M	3.4M	-	-	0.0	-	-
-	100	30.6G	806M	15.0M	3.9K	86.7	-	0.0

Workload RPS / Latency - 'g': more graphs, 't/T': change timescale



Management logs

```
[15:07:05 rd-oomd] [../src/oomd/config/ConfigTypes.cpp:86]        cgroup=workload.slice/
*,sideload.slice/*,system.slice/*
[15:07:05 rd-oomd] [../src/oomd/Oomd.cpp:95] Running oomd
[15:07:46 rd-sideloader] UNKNOWN
[15:12:17 rd-agent] [INFO] hashd: Updating "hashd-A" to lat=100.00ms@95.00% rps=12410 mem=33.84%
log=1.05Mbps frac=1.00
[15:12:17 rd-agent] [INFO] svc: "rd-hashd-A.service" started (Running)
[15:12:42 rd-sideloader] OVERLOAD: cpu margin 7.84 is too low, hold=0s
```

Other logs

```
[15:07:04 rd-iocost-bench] io.cost.model: 8:64 rbps=323298014 rseqiops=38819 rrandiops=43230
wbps=355938482 wseqiops=28836 wrandiops=29095
[15:07:04 rd-iocost-bench] io.cost.qos: 8:64 rpct=95 rlat=3416 wpct=95 wlat=4301 min=60 max=100
[15:07:04 init.scope] rd-iocost-bench.service: Succeeded.
[15:07:04 init.scope] Stopped rd-iocost-bench.service /var/lib/resctl-demo/misc-bin/
iocost_coef_gen.py --json /var/lib/resctl-demo/scratch/iocost-coef/iocost-coef.json --testfile-
dev sde --duration 60.
[15:07:04 init.scope] rd-iocost-bench.service: Consumed 6min 49.894s CPU time.
```

[comp.cgroup] Cgroup and Resource Protection - 'i': index, 'b': back

Cgroup and Resource Protection

Scenario: A web server, an external memory leak, and no resource control

Imagine a fleet of web servers whose main job is running a web application to service user requests. But like all systems, a lot of other apps need to run: The system software and the web app require regular updates; fleet-wide maintenance tools keep the configuration in compliance; various monitoring and alarm frameworks need to run - and so on.

Let's say a maintenance program started by cron is malfunctioning and keeps leaking memory, and that the malfunction is wall-clock dependent: This is a bad scenario that would stay hidden during testing, and then trigger at the same time in production. What would happen to the system and fleet without resource control enabled?

We can simulate this scenario with the rd-hashd workload simulator, and a test program intentionally designed to leak memory. rd-hashd is already started targeting the full load. Give it some time to ramp up, and let its memory footprint grow to closely fill up the machine. Check out the memory utilization graph by pressing 'g'.

Once the workload sufficiently ramps up, you can see how the system behaves without any protection, by selecting the button below to disable all resource control mechanisms and start a memory-hog program. The problematic program will start as rd-sysload-memory-hog.service under system:

[Disable all resource control features and start memory hog]

WARNING: Because the system is running without any protection, nothing can guarantee the system's responsiveness. Everything, including this demo program, will get sluggish, and might completely stall. Depending on how the kernel OOM killer reacts, the system may or may not recover in a reasonable amount of time. To avoid the need for a reboot, stop the experiment once the system becomes sluggish:

[Stop the memory hog and restore resource control]

That wasn't ideal, was it? If this were a production environment and the failure happened across a large number of machines at the same time, our users would notice the disturbances, and the whole site might go down.

The components

- rd-hashd: Latency sensitive main workload
- Sysloads: Possibly misbehaving secondary workloads
- Sideloads: Secondary workloads running under sideloader
- rd-agent: System and experiment management
- resctl-demo: TUI for doc, monitoring and contol
- resctl-bench: Whole system benchmarks using the above.

[intro.sysreqs] System Requirements - 'i': index, 'b': back

* SysReq::IoCost: blk-iocost is the new IO controller which can comprehensively control IO capacity distribution proportionally. Enabled with CONFIG_BLK_CGROUP_IOCOST. For details: https://lwn.net/Articles/793460/

* SysReq::IoCostVer: blk-iocost received significant updates to improve control quality and visibility during the v5.10 development cycle. A kernel with these updates is recommended. For details: https://lwn.net/Articles/830397/

* SysReq::NoOtherIoControllers: Other IO controllers - io.max and io.latency - can interfere and shouldn't have active configurations.

  If configured through systemd, remove all IO{Read|Write}{Bandwidth|IOPS}Max and IoDeviceLatencyTargetSec configurations.

* SysReq::AnonBalance: Kernel memory management received a major update during the v5.8 development cycle which put anonymous memory on an equal footing with page cache and made swap useful, especially on SSDs. For details: https://lwn.net/Articles/821105/

* SysReq::Btrfs: Working IO isolation requires support from filesystem to avoid priority inversions. Currently, btrfs is the only supported filesystem.

  The OS must be installed with btrfs as the root filesystem.

* SysReq::BtrfsAsyncDiscard: Many SSDs show significant latency spikes when discards are issued in bulk, which can lead to severe priority inversions. Async discard is a btrfs feature that paces and reduces the total amount of discards.

  It can be enabled with "discard=async" mount option on kernels >= v5.6. If available, resctl-demo will automatically remount the filesystem with the mount option. For details: https://lwn.net/Articles/805300/

* SysReq::NoCompositeStorage: Currently, composite block devices, such as dm and md, break the chain of custody for IOs, allowing cgroups to escape IO control and cause severe priority inversions.

  The filesystem must be on a physical device.

* SysReq::IoSched: bfq IO scheduler's implementation of proportional IO

Protection

Scenario

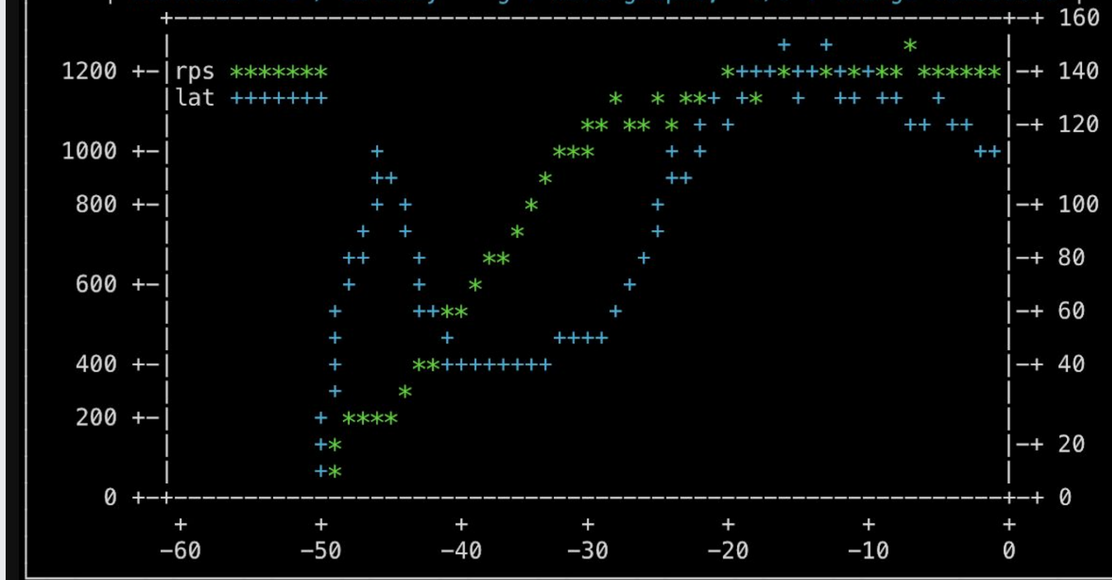
- Web server is running full-tilt
- A management job running from chef has a new bug leaking memory.
- Can we survive if the same behavior happens across the fleet at the same time?

Facebook Resource Control Demo v1.0.0 - 'q': quit

```
[  Running  ] 2021-01-22 04:08:25 PM
[  config   ] satisfied: 18  missed:  0
[   oomd    ] workload: +pressure -senpai  system: +pressure -senpai
[ sideload  ] jobs:  0/ 0  failed:  0  cfg_warn:  0  +overload -crit
[  sysload  ] jobs:  0/ 0  failed:  0
[ workload  ] load:  100%  lat: 116ms  cpu: 91.8%  mem: 25.3G  io: 27.2M
```

	cpu%	mem	swap	rbps	wbps	cpuP%	memP%	ioP%
workload	92.3	25.7G	144K	34.0M	4.0K	91.2	-	0.0
sideload	-	-	-	-	-	-	-	-
hostcritical	7.4	744M	-	-	19.7K	34.0	-	0.1
system	0.0	17.7M	1.5M	-	-	0.0	-	-
user	0.0	517M	3.4M	-	-	0.0	-	-
-	100	27.3G	678M	34.0M	23.7K	93.0	-	0.0

Workload RPS / Latency - 'g': more graphs, 't/T': change timescale



Management logs

```
workload.slice
[16:01:39 rd-sideloader] OVERLOAD: cpu margin 0.00 is too low, hold=0s
[16:01:42 rd-agent] [INFO] svc: "rd-hashd-A.service" started (Running)
[16:01:58 rd-sideloader] OVERLOAD: end, resuming normal operation
[16:02:09 rd-sideloader] OVERLOAD: cpu margin 5.66 is too low, hold=5s
[16:03:08 rd-agent] [INFO] svc: "rd-hashd-A.service" stopped (NotFound)
[16:03:24 rd-sideloader] OVERLOAD: end, resuming normal operation
[16:07:29 rd-agent] [INFO] svc: "rd-hashd-A.service" started (Running)
[16:07:56 rd-sideloader] OVERLOAD: cpu margin 5.21 is too low, hold=0s
```

Other logs

```
[15:07:04 rd-iocost-bench] io.cost.model: 8:64 rbps=323298014 rseqiops=38819
rrandiops=43230 wbps=355938482 wseqiops=28836 wrandiops=29095
[15:07:04 rd-iocost-bench] io.cost.qos: 8:64 rpct=95 rlat=3416 wpct=95
wlat=4301 min=60 max=100
[15:07:04 init.scope] rd-iocost-bench.service: Succeeded.
[15:07:04 init.scope] Stopped rd-iocost-bench.service /var/lib/resctl-demo/
misc-bin/iocost_coef_gen.py --json /var/lib/resctl-demo/scratch/iocost-coef/
iocost-coef.json --testfile-dev sde --duration 60.
[15:07:04 init.scope] rd-iocost-bench.service: Consumed 6min 49.894s CPU time.
```

[comp.cgroup] Cgroup and Resource Protection - 'i': index, 'b': back

Cgroup and Resource Protection

Scenario: A web server, an external memory leak, and no resource control

Imagine a fleet of web servers whose main job is running a web application to service user requests. But like all systems, a lot of other apps need to run: The system software and the web app require regular updates; fleet-wide maintenance tools keep the configuration in compliance; various monitoring and alarm frameworks need to run - and so on.

Let's say a maintenance program started by cron is malfunctioning and keeps leaking memory, and that the malfunction is wall-clock dependent: This is a bad scenario that would stay hidden during testing, and then trigger at the same time in production. What would happen to the system and fleet without resource control enabled?

We can simulate this scenario with the rd-hashd workload simulator, and a test program intentionally designed to leak memory. rd-hashd is already started targeting the full load. Give it some time to ramp up, and let its memory footprint grow to closely fill up the machine. Check out the memory utilization graph by pressing 'g'.

Once the workload sufficiently ramps up, you can see how the system behaves without any protection, by selecting the button below to disable all resource control mechanisms and start a memory-hog program. The problematic program will start as rd-sysload-memory-hog.service under system:

[Disable all resource control features and start memory hog]

WARNING: Because the system is running without any protection, nothing can guarantee the system's responsiveness. Everything, including this demo program, will get sluggish, and might completely stall. Depending on how the kernel OOM killer reacts, the system may or may not recover in a reasonable amount of time. To avoid the need for a reboot, stop the experiment once the system becomes sluggish:

[Stop the memory hog and restore resource control]

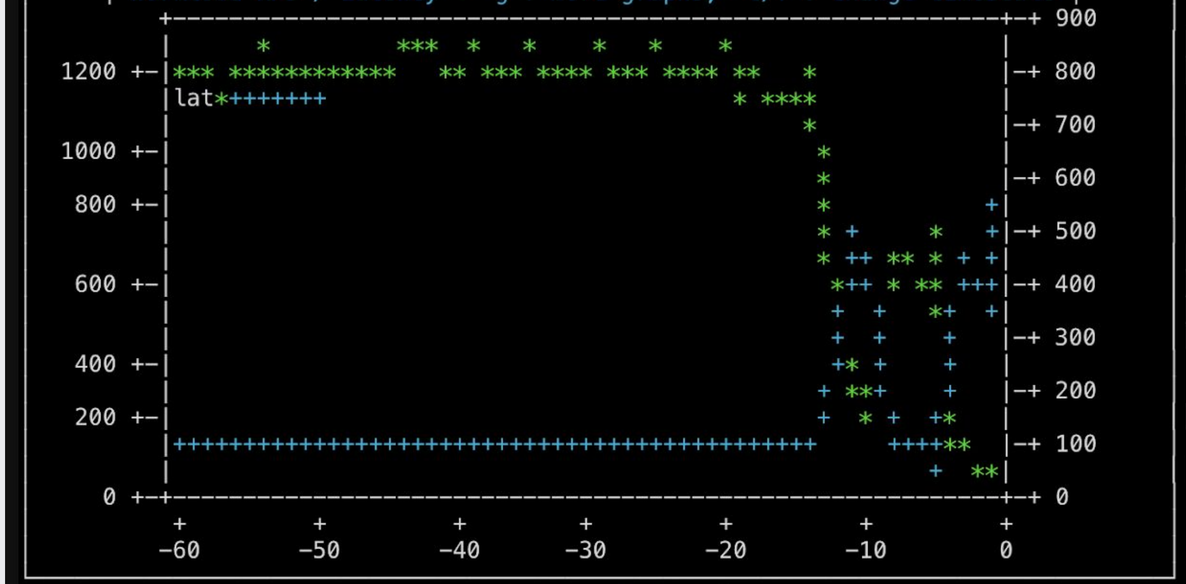
That wasn't ideal, was it? If this were a production environment and the failure happened across a large number of machines at the same time, our users would notice the disturbances, and the whole site might go down.

Facebook Resource Control Demo v1.0.0 - 'q': quit

```
[  Running  ] 2021-01-22 04:19:49 PM
[  config   ] satisfied: 18  missed:  0, -cpu -mem -io
[   oomd    ]
[ sideload  ]
[  sysload  ] jobs:  1/ 1  failed:  0
[ workload  ] load:  5.3%  lat: 832ms  cpu:  4.5%  mem: 24.0G  io:  268M
```

	cpu%	mem	swap	rbps	wbps	cpuP%	memP%	ioP%
workload	4.2	24.3G	752M	63.5M	193M	0.2	66.3	66.3
sideload	-	-	-	-	-	-	-	-
hostcritical	5.7	707M	44.0K	-	-	1.3	-	0.0
system	0.9	5.7G	1.0G	-	34.2M	0.0	-	-
user	0.0	85.6M	3.1M	-	-	0.0	-	-
-	14.2	31.0G	1.8G	63.7M	227M	1.4	63.2	60.3

Workload RPS / Latency - 'g': more graphs, 't/T': change timescale



Management logs

```
[16:19:28 rd-agent] [INFO] svc: "rd-sideloader.service" stopped (NotFound)
[16:19:28 rd-agent] [INFO] resctl: Controller enable state disagrees with
overrides, fixing
[16:19:29 rd-agent] [INFO] resctl: "/sys/fs/cgroup/workload.slice/memory.low"
should be "0" but is "23068672", fixing
[16:19:29 rd-agent] [INFO] svc: "rd-sysload-memory-hog.service" started
(Running)
[16:19:29 rd-agent] [INFO] resctl: Controller enable state disagrees with
overrides, fixing
```

Other logs

```
[16:19:40 rd-sysload-memory-hog] size=3.98G rbps=0.00M wbps=322.80M
[16:19:41 rd-sysload-memory-hog] size=4.32G rbps=0.00M wbps=341.59M
[16:19:42 rd-sysload-memory-hog] size=4.67G rbps=0.00M wbps=339.08M
[16:19:43 rd-sysload-memory-hog] size=5.01G rbps=0.00M wbps=337.66M
[16:19:44 rd-sysload-memory-hog] size=5.36G rbps=0.00M wbps=338.54M
[16:19:45 rd-sysload-memory-hog] size=5.72G rbps=0.00M wbps=343.02M
[16:19:46 rd-sysload-memory-hog] size=6.05G rbps=0.00M wbps=339.83M
[16:19:47 rd-sysload-memory-hog] size=6.40G rbps=0.00M wbps=338.81M
[16:19:48 rd-sysload-memory-hog] size=6.75G rbps=0.00M wbps=339.97M
```

[comp.cgroup] Cgroup and Resource Protection - 'i': index, 'b': back

Cgroup and Resource Protection

Scenario: A web server, an external memory leak, and no resource control

Imagine a fleet of web servers whose main job is running a web application to service user requests. But like all systems, a lot of other apps need to run: The system software and the web app require regular updates; fleet-wide maintenance tools keep the configuration in compliance; various monitoring and alarm frameworks need to run - and so on.

Let's say a maintenance program started by cron is malfunctioning and keeps leaking memory, and that the malfunction is wall-clock dependent: This is a bad scenario that would stay hidden during testing, and then trigger at the same time in production. What would happen to the system and fleet without resource control enabled?

We can simulate this scenario with the rd-hashd workload simulator, and a test program intentionally designed to leak memory. rd-hashd is already started targeting the full load. Give it some time to ramp up, and let its memory footprint grow to closely fill up the machine. Check out the memory utilization graph by pressing 'g'.

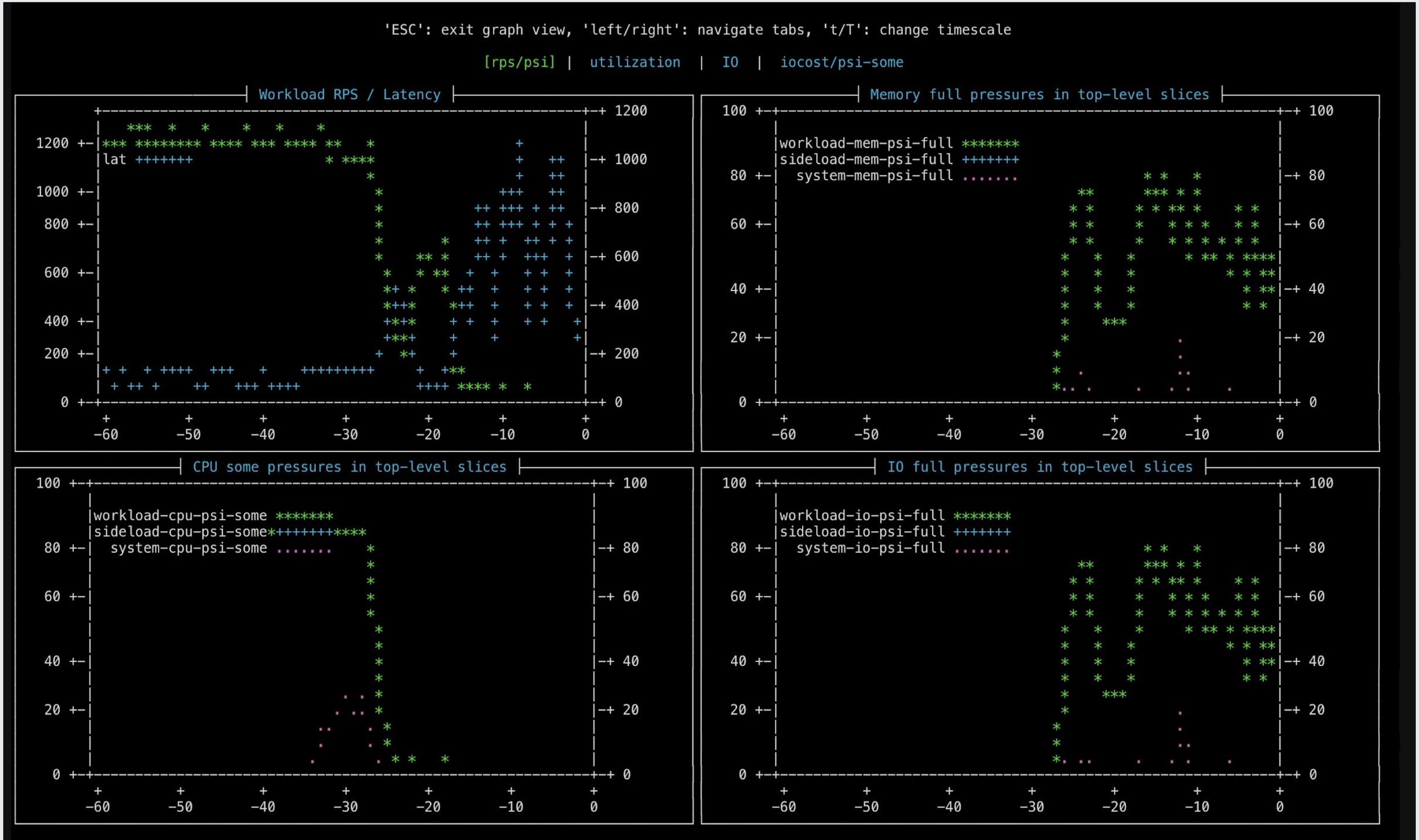
Once the workload sufficiently ramps up, you can see how the system behaves without any protection, by selecting the button below to disable all resource control mechanisms and start a memory-hog program. The problematic program will start as rd-sysload-memory-hog.service under system:

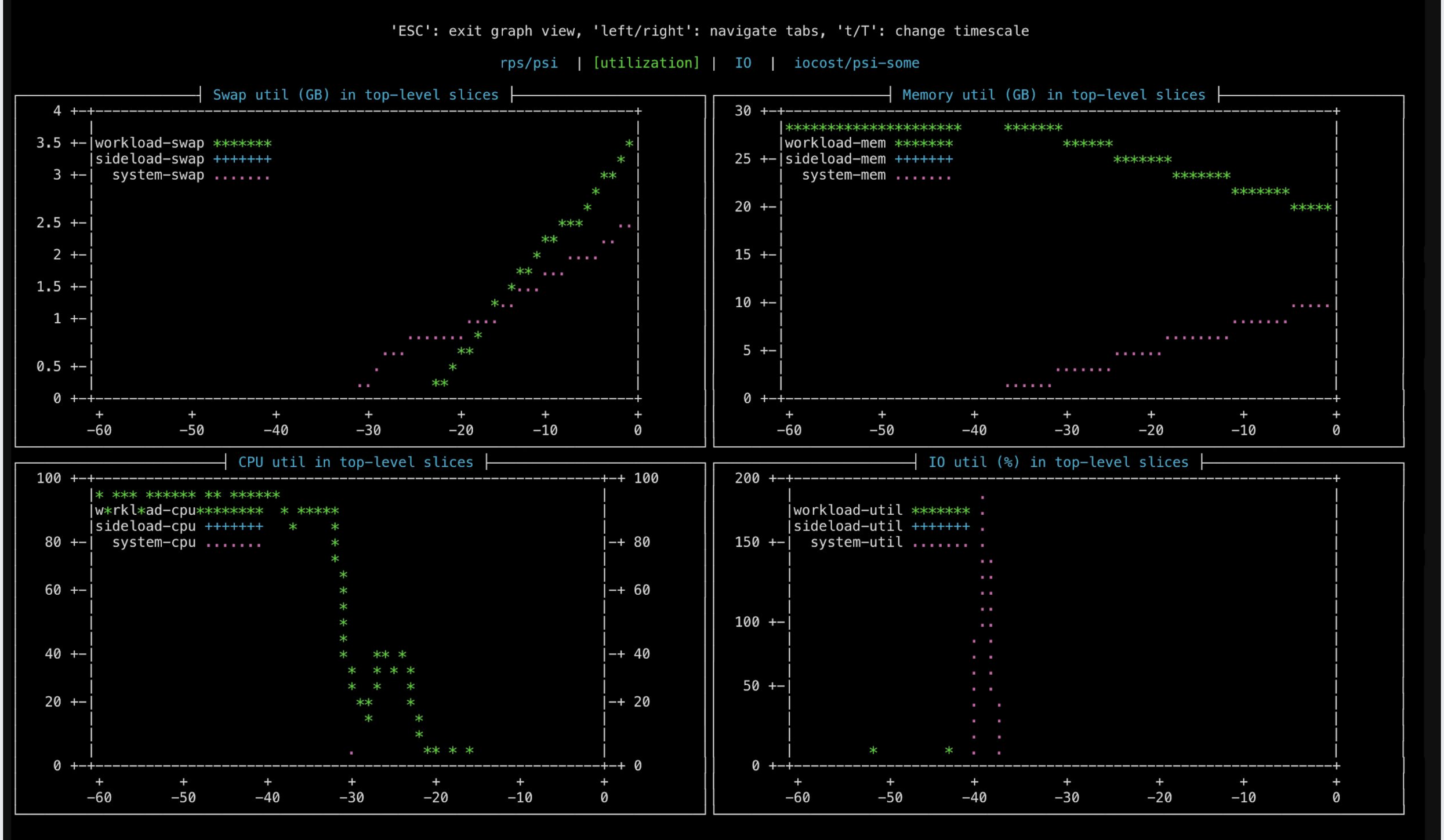
[Disable all resource control features and start memory hog]

WARNING: Because the system is running without any protection, nothing can guarantee the system's responsiveness. Everything, including this demo program, will get sluggish, and might completely stall. Depending on how the kernel OOM killer reacts, the system may or may not recover in a reasonable amount of time. To avoid the need for a reboot, stop the experiment once the system becomes sluggish:

[Stop the memory hog and restore resource control]

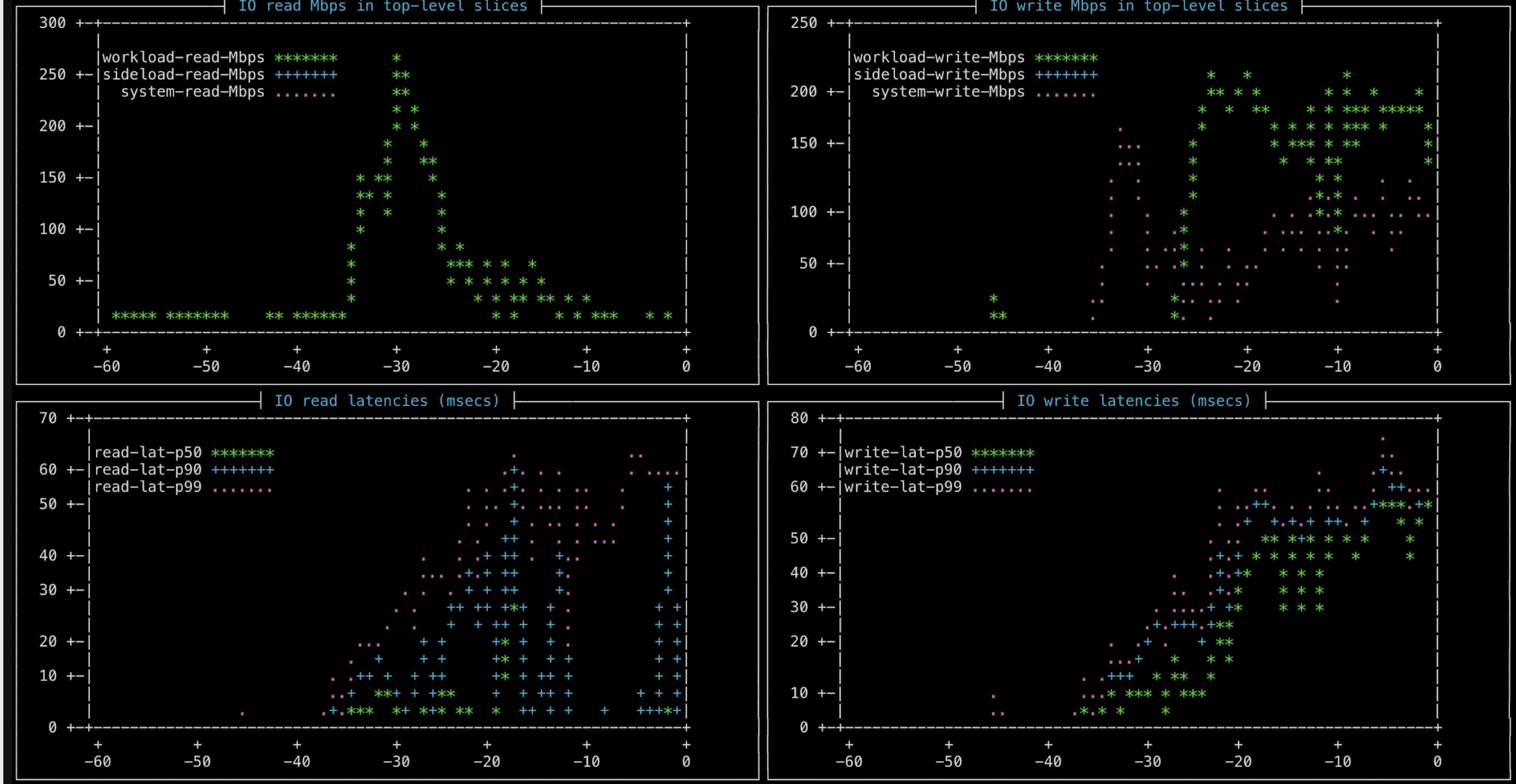
That wasn't ideal, was it? If this were a production environment and the failure happened across a large number of machines at the same time, our users would notice the disturbances, and the whole site might go down.


'ESC': exit graph view, 'left/right': navigate tabs, 't/T': change timescale
[rps/psi] | utilization | IO | iocost/psi-some
Workload RPS / Latency
Memory full pressures in top-level slices
workload-mem-psi-full
sideload-mem-psi-full
system-mem-psi-full
CPU some pressures in top-level slices
workload-cpu-psi-some
sideload-cpu-psi-some
system-cpu-psi-some
IO full pressures in top-level slices
workload-io-psi-full
sideload-io-psi-full
system-io-psi-full
lat
-60
-50
-40
-30
-20
-10
0


'ESC': exit graph view, 'left/right': navigate tabs, 't/T': change timescale
rps/psi | [utilization] | IO | iocost/psi-some
Swap util (GB) in top-level slices
workload-swap
sideload-swap
system-swap
Memory util (GB) in top-level slices
workload-mem
sideload-mem
system-mem
CPU util in top-level slices
workload-cpu
sideload-cpu
system-cpu
IO util (%) in top-level slices
workload-util
sideload-util
system-util

'ESC': exit graph view, 'left/right': navigate tabs, 't/T': change timescale

rps/psi | utilization | [IO] | iocost/psi-some



Same scenario with protection

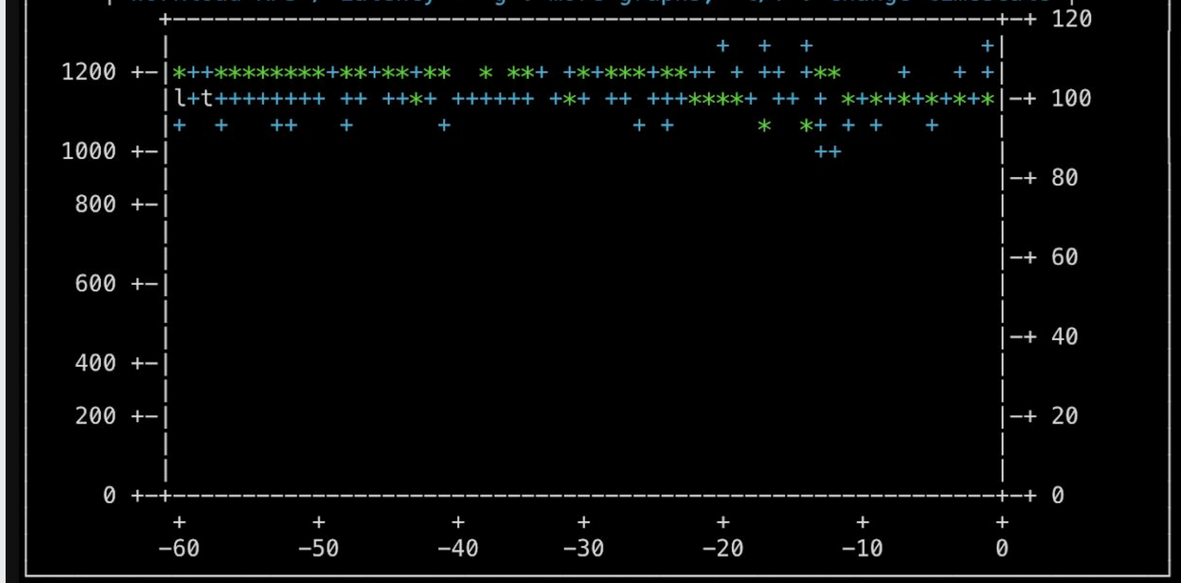
- cpu.weight, memory.low and io.weight configured
- oomd on watch

┤ Facebook Resource Control Demo v1.0.0 - 'q': quit ├

```
[  Running  ] 2021-01-22 04:34:25 PM
[  config   ] satisfied: 18  missed:  0
[   oomd    ] workload: +pressure -senpai  system: +pressure -senpai
[ sideload  ] jobs:  0/ 0  failed:  0  cfg_warn:  0  +overload -crit
[  sysload  ] jobs:  1/ 1  failed:  0
[ workload  ] load: 80.8%  lat: 115ms  cpu: 89.2%  mem: 27.9G  io:  100M
```

	cpu%	mem	swap	rbps	wbps	cpuP%	memP%	ioP%
workload	87.1	27.9G	2.5G	111M	10.3M	65.0	2.8	3.4
sideload	-	-	-	-	-	-	-	-
hostcritical	6.6	721M	-	-	-	18.7	-	0.0
system	0.0	1.6G	345M	10.6K	38.0M	0.3	84.1	84.1
user	0.0	84.8M	21.2M	-	-	0.0	-	-
-	97.1	31.0G	7.2G	111M	48.1M	67.7	4.2	2.0

┤ Workload RPS / Latency - 'g': more graphs, 't/T': change timescale ├



┤ Management logs ├

```
[16:33:12 rd-agent] [INFO] resctl: "/sys/fs/cgroup/init.scope/cpu.weight"
should be 10 but is Ok(100), fixing
[16:33:12 rd-agent] [INFO] resctl: "/sys/fs/cgroup/hostcritical.slice/
cpu.weight" should be 10 but is Ok(100), fixing
[16:33:12 rd-sideloader] INIT: sideloads in sideload.slice, main workloads in
workload.slice
[16:33:12 rd-sideloader] OVERLOAD: cpu margin 0.00 is too low, hold=0s
[16:34:03 rd-agent] [INFO] svc: "rd-sysload-memory-hog.service" started
(Running)
```

┤ Other logs ├

```
[16:34:10 rd-sysload-memory-hog] size=1.59G rbps=0.00M wbps=0.11M
[16:34:12 rd-sysload-memory-hog] size=1.59G rbps=0.00M wbps=0.18M
[16:34:13 rd-sysload-memory-hog] size=1.60G rbps=0.00M wbps=2.66M
[16:34:14 rd-sysload-memory-hog] size=1.60G rbps=0.00M wbps=0.81M
[16:34:15 rd-sysload-memory-hog] size=1.60G rbps=0.00M wbps=0.24M
[16:34:16 rd-sysload-memory-hog] size=1.84G rbps=0.00M wbps=247.22M
[16:34:19 rd-sysload-memory-hog] size=1.90G rbps=0.00M wbps=22.16M
[16:34:24 rd-sysload-memory-hog] size=1.90G rbps=0.00M wbps=0.02M
[16:34:25 rd-sysload-memory-hog] size=1.90G rbps=0.00M wbps=0.12M
```

┤ [comp.cgroup] Cgroup and Resource Protection - 'i': index, 'b': back ├

slices.

You can check out what cgroups are on your system by running `systemd-cgls`. The kernel interface is a pseudo filesystem mounted under /sys/fs/cgroup.

Turning on resource protection

Once the kernel understands what applications are on the system and how they're grouped in cgroups, it can monitor resource consumption and control distribution along that hierarchy.

We'll repeat the same test run we did above, but leaving all the resource protection configurations turned on.

rd-hashd should already be running at full tilt. Once workload's memory usage stops growing, let's start the same memory hog, but without touching anything else:

[Start memory hog]

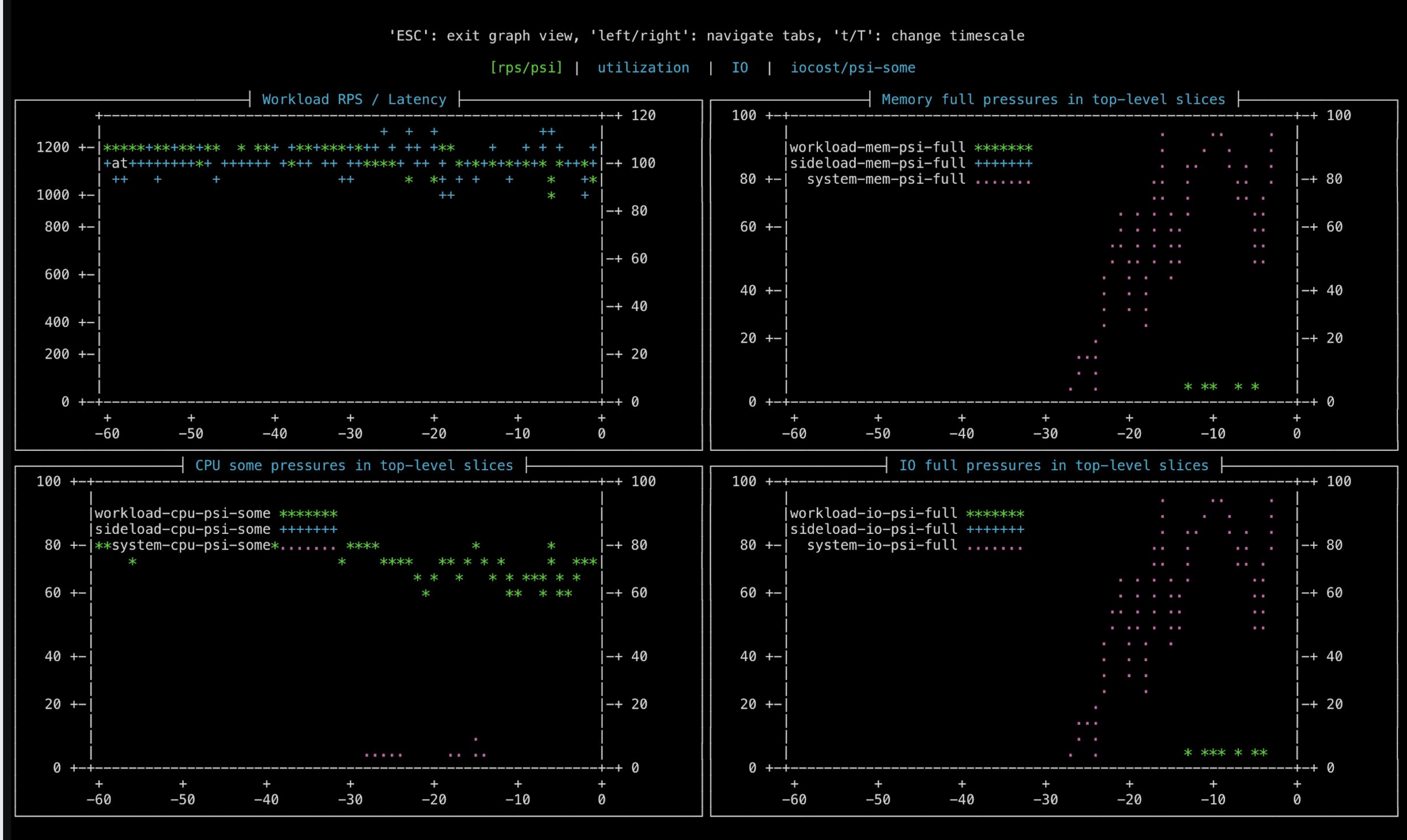
Monitor the RPS and other resource consumption in the graph view ('g'). The RPS may go down a little and dip occasionally, but it'll stay close to full capacity no matter how long you let the memory hog go on. Eventually the memory hog will be killed off by OOMD. Try it multiple times by clearing the memory hog with the following button, and restarting it with the start button above:

[Stop memory hog]

In this scenario, with resource control on, our site's not going down, and users probably aren't even noticing. If we have a working monitoring mechanism, the OOMD kills will raise an alarm, people can quickly find out which program was malfunctioning, and hunt down the bug. What could have been a site-wide outage got de-escalated into an internal oops.

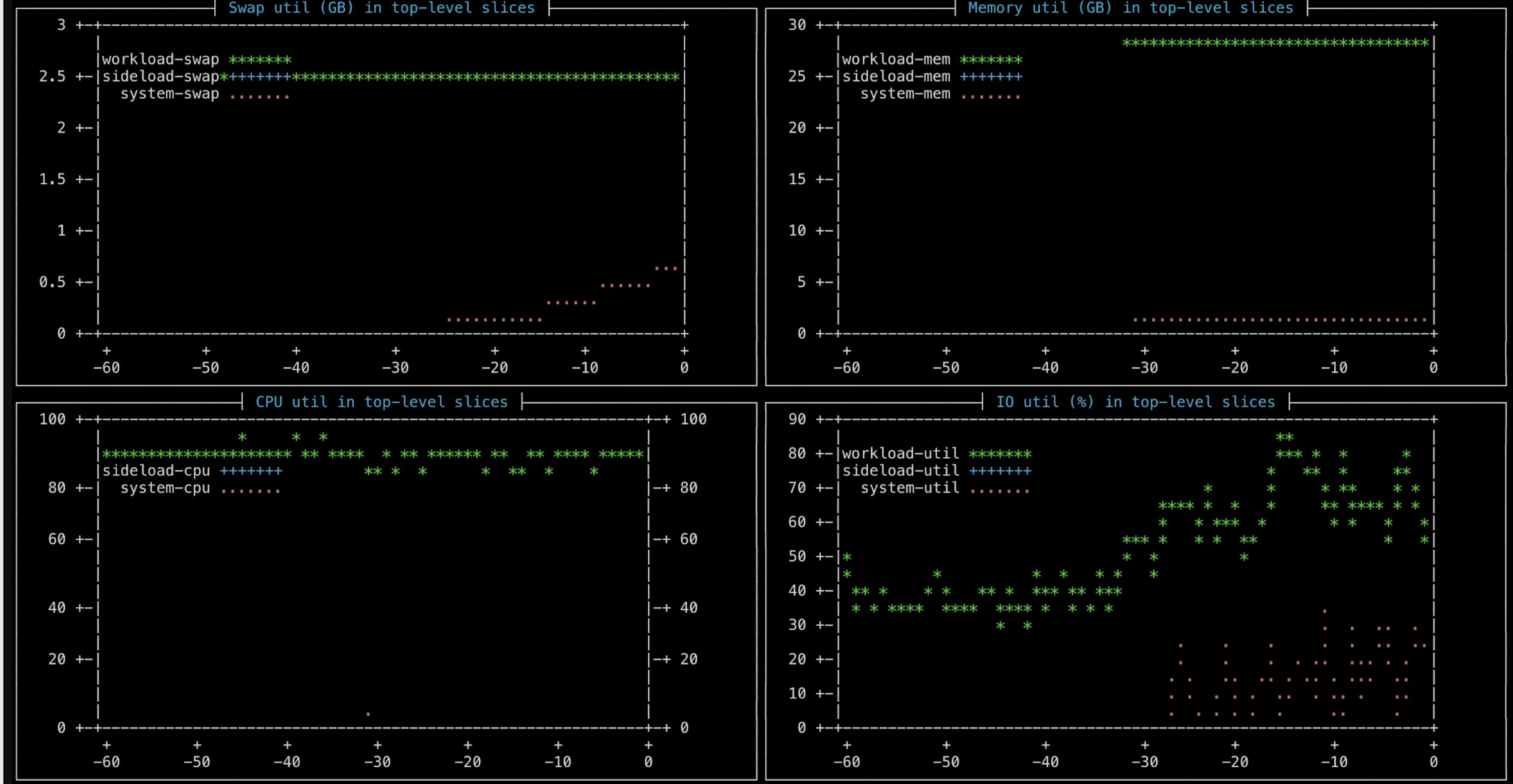
This is cgroup resource protection at work. The kernel understood which part was important and protected its resources at the expense of the memory hog running in the low-priority portion of the system. Eventually, the situation for the memory hog became unsustainable and OOMD terminated it.

We'll go into details in the following pages but here's a brief summary of how resource protection is configured in this demo.


'ESC': exit graph view, 'left/right': navigate tabs, 't/T': change timescale
[rps/psi] | utilization | IO | iocost/psi-some
Workload RPS / Latency
Memory full pressures in top-level slices
workload-mem-psi-full
sideload-mem-psi-full
system-mem-psi-full
CPU some pressures in top-level slices
workload-cpu-psi-some
sideload-cpu-psi-some
system-cpu-psi-some
IO full pressures in top-level slices
workload-io-psi-full
sideload-io-psi-full
system-io-psi-full
-60
-50
-40
-30
-20
-10
0

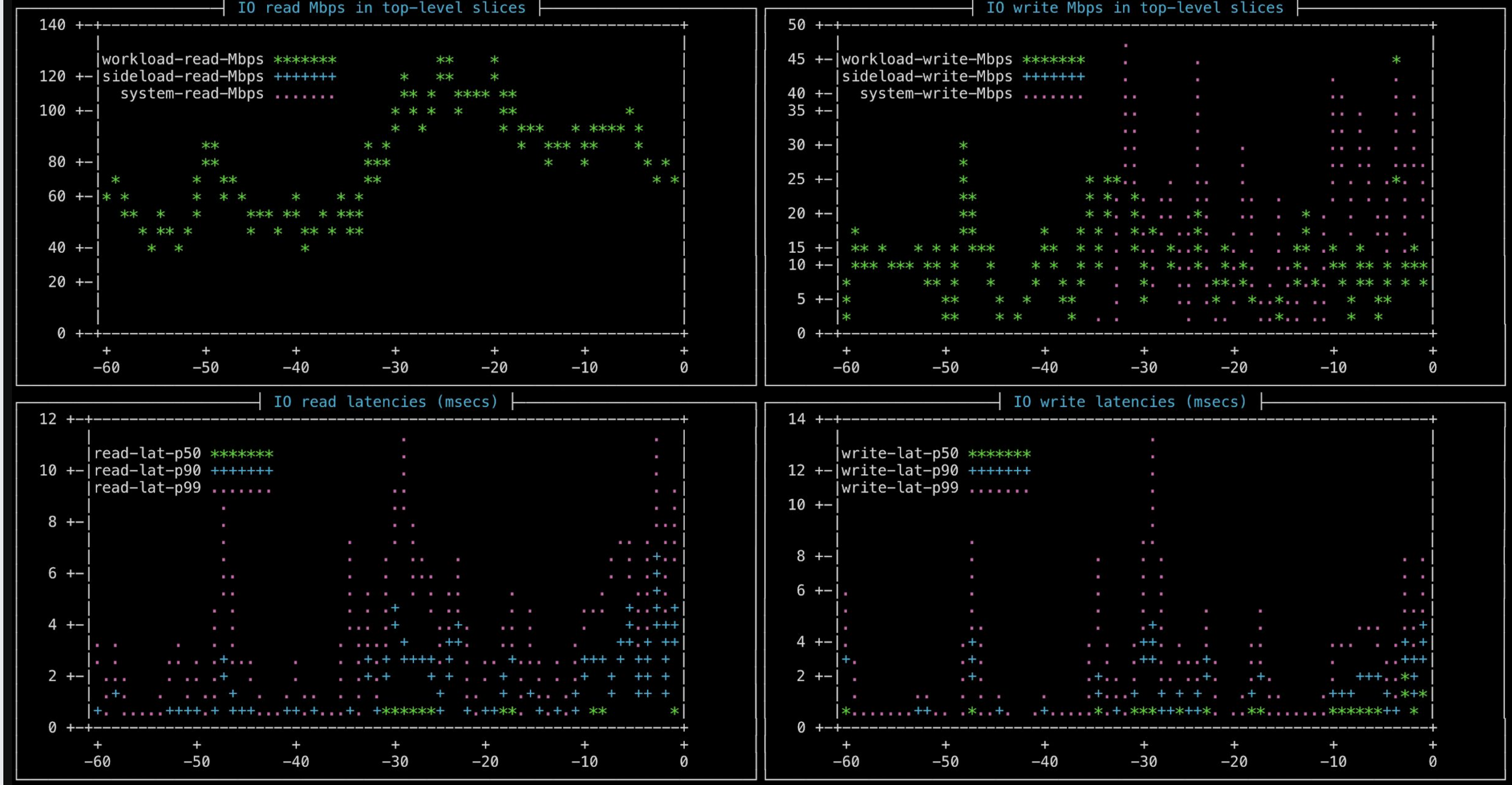
'ESC': exit graph view, 'left/right': navigate tabs, 't/T': change timescale

rps/psi | [utilization] | IO | iocost/psi-some



'ESC': exit graph view, 'left/right': navigate tabs, 't/T': change timescale

rps/psi | utilization | [IO] | iocost/psi-some



```
[Jan 22 16:34:53 rd-oomd] [../src/oomd/plugins/BaseKillPlugin.cpp:319] Killed 1: 112038(memory-growth.p)
[Jan 22 16:34:53 rd-oomd] [../src/oomd/plugins/BaseKillPlugin.cpp:319] Killed 1: 112038(memory-growth.p)
[Jan 22 16:34:54 rd-oomd] [../src/oomd/Log.cpp:117] 36.84 22.28 6.04 system.slice/rd-sysload-memory-hog.service 1058918400
ruleset:[protection against heavy system.slice thrashing] detectorgroup:[Sustained thrashing in system.slice]
killer:kill_by_memory_size_or_growth v2
```

Sideloading

What’s sideloading?

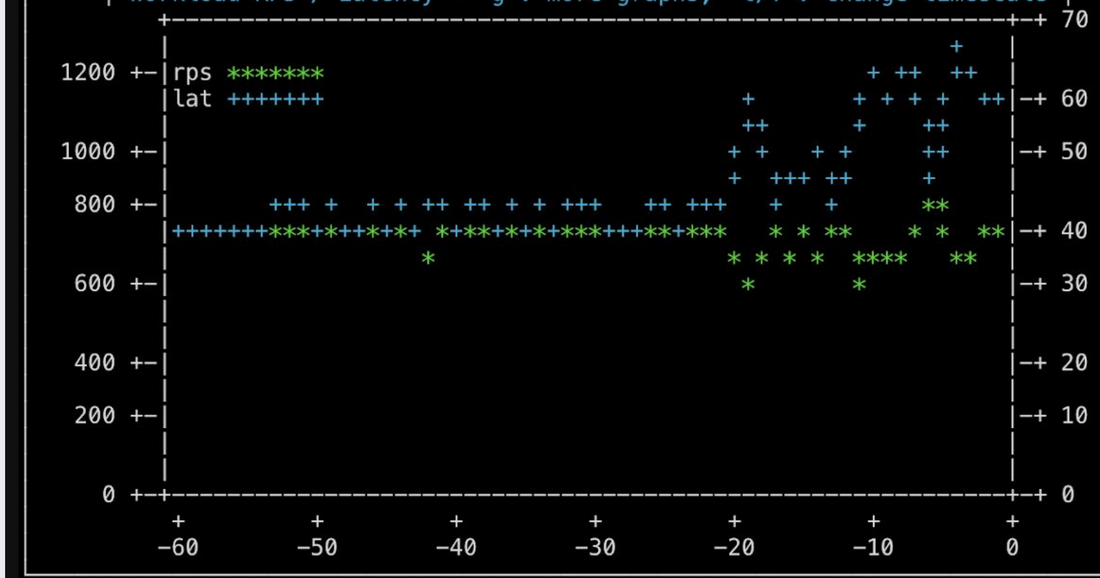
- Machines can’t be utilized fully for reasons such as scheduling inefficiency and DR buffer
- Maybe simply a generalized case of protection?

Facebook Resource Control Demo v1.0.0 - 'q': quit

```
[  Running  ] 2021-01-22 04:53:42 PM
[  config   ] satisfied: 18  missed:  0
[   oomd    ] workload: +pressure -senpai  system: +pressure -senpai
[ sideload  ] jobs:  0/ 0  failed:  0  cfg_warn:  0  +overload -crit
[  sysload  ] jobs:  1/ 1  failed:  0
[ workload  ] load: 56.2%  lat:  61ms  cpu: 54.1%  mem: 25.2G  io:  2.7M
```

	cpu%	mem	swap	rbps	wbps	cpuP%	memP%	ioP%
workload	49.7	25.7G	772K	1.6M	4.0K	35.8	-	0.1
sideload	-	-	-	-	-	-	-	-
hostcritical	6.4	741M	32.0K	-	-	10.6	-	0.0
system	43.4	2.9G	792K	32.0K	143K	50.5	-	0.0
user	0.1	84.9M	21.0M	-	-	0.2	-	-
-	100	30.0G	578M	1.6M	5.5M	67.0	-	0.0

Workload RPS / Latency - 'g': more graphs, 't/T': change timescale



Management logs

```
[16:49:10 rd-agent] [INFO] hashd: Updating "hashd-A" to lat=100.00ms@95.00%
rps=745 mem=33.84% log=1.05Mbps frac=1.00
[16:49:10 rd-agent] [INFO] svc: "rd-hashd-A.service" started (Running)
[16:49:18 rd-sideloader] OVERLOAD: end, resuming normal operation
[16:52:47 rd-agent] [INFO] hashd: Updating "hashd-A" to lat=200.00ms@95.00%
rps=745 mem=33.84% log=1.05Mbps frac=1.00
[16:52:47 rd-agent] [INFO] svc: "rd-sysload-compile-job.service" started
(Running)
[16:53:26 rd-sideloader] OVERLOAD: cpu margin 8.06 is too low, hold=0s
```

Other logs

```
[16:53:43 rd-sysload-compile-job] UNKNOWN
[16:53:43 rd-sysload-compile-job]   CC      arch/x86/events/core.o
[16:53:43 rd-sysload-compile-job]   HDRTEST usr/include/drm/vc4_drm.h
[16:53:43 rd-sysload-compile-job]   AS [M]  arch/x86/crypto/serpent-avx-x86_64-
asm_64.o
[16:53:43 rd-sysload-compile-job]   CC [M]  arch/x86/crypto/serpent_avx_glue.o
[16:53:43 rd-sysload-compile-job]   HDRTEST usr/include/drm/virtgpu_drm.h
[16:53:43 rd-sysload-compile-job]   CC      arch/x86/entry/syscall_64.o
[16:53:43 rd-sysload-compile-job]   AR      arch/x86/kernel/acpi/built-in.a
```

[side.intro] What is Sideloading? - 'i': index, 'b': back

A naive approach

In the previous chapter, we demonstrated that resource control can protect the main workload from the rest of the system. While rd-hashd was running at full load, we could throw many types of misbehaving workloads at the system with only limited impact on the main workload. We also saw that CPU control couldn't protect rd-hashd's latency from a CPU hog when rd-hashd was running close to full load.

If the CPU control can protect the main workload at moderate load levels, maybe we can simply run the sideloads under the same resource control configuration and stop them when the main workload's load spikes. Let's see how latency at 60% load and ramping up from there are impacted by running sideloads this way.

rd-hashd should already be running at 60% load. Once it's warmed up, start a Linux build job with 1x CPU count concurrency. Depending on the hardware and kernel configurations, rd-hashd may fail to hold the RPS with the usual 100ms latency target, so let's relax the latency target to 200ms too. Pay attention to how the latency in the left graph pane changes:

[Relax rd-hashd latency target and start linux build job]

Note how RPS is holding but latency deteriorates sharply. Press 'g' and check out the resource pressure graphs. Even though this page set the CPU weight of the build job only at a hundredth of rd-hashd, CPU pressure is noticeable. We'll get back to this later.

Now let's push the load up to 100% and see whether its ability to ramp up is impacted too:

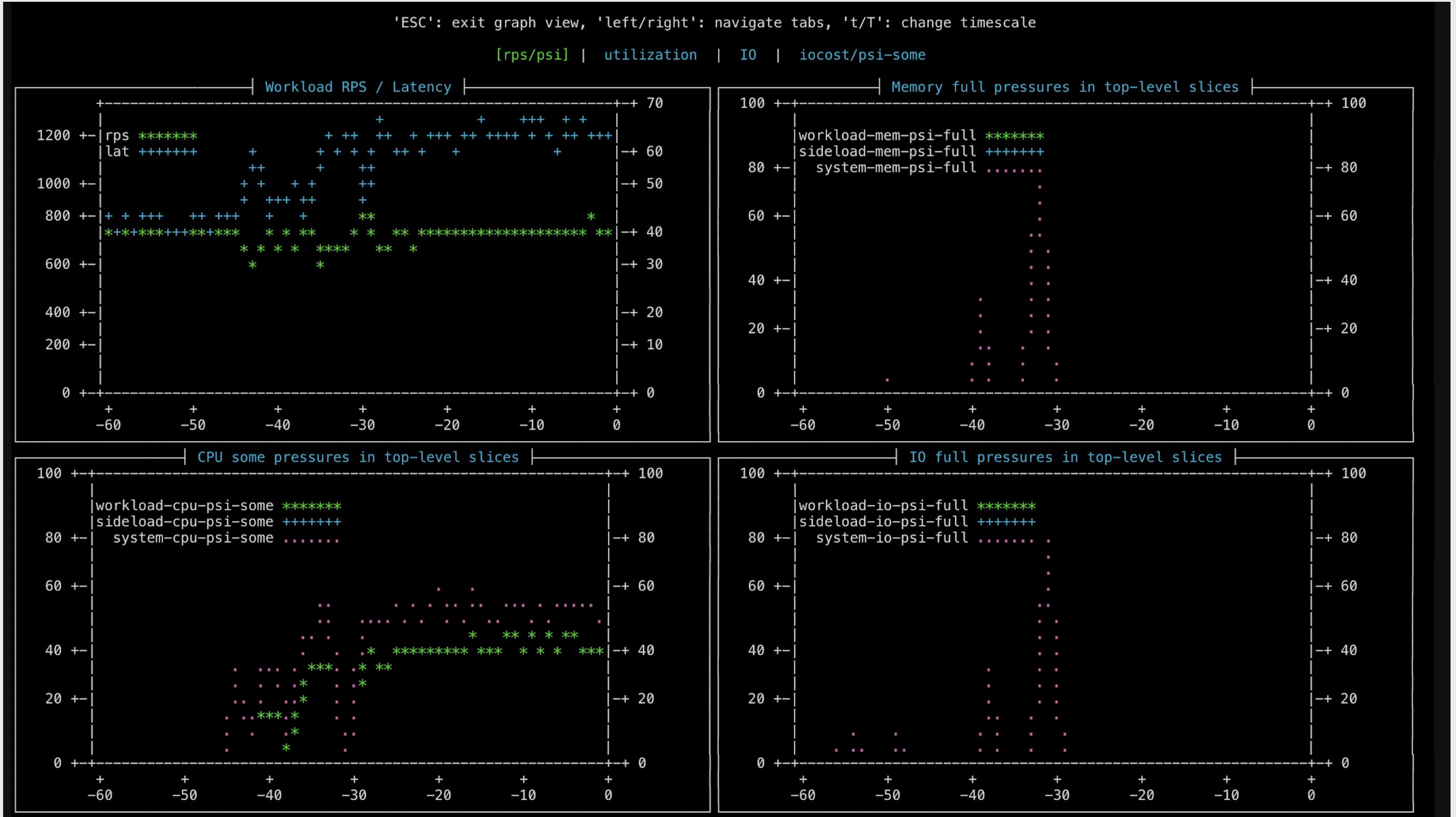
[Set full load]

It climbs, but seems kind of sluggish. Let's compare it with a load rising but without the build job:

[Stop linux build, reset latency target and set 60% load]

Wait until it stabilizes, then ramp it up to 100%:

[Set full load]


'ESC': exit graph view, 'left/right': navigate tabs, 't/T': change timescale
[rps/psi] | utilization | IO | iocost/psi-some
Workload RPS / Latency
rps
lat
Memory full pressures in top-level slices
workload-mem-psi-full
sideload-mem-psi-full
system-mem-psi-full
CPU some pressures in top-level slices
workload-cpu-psi-some
sideload-cpu-psi-some
system-cpu-psi-some
IO full pressures in top-level slices
workload-io-psi-full
sideload-io-psi-full
system-io-psi-full

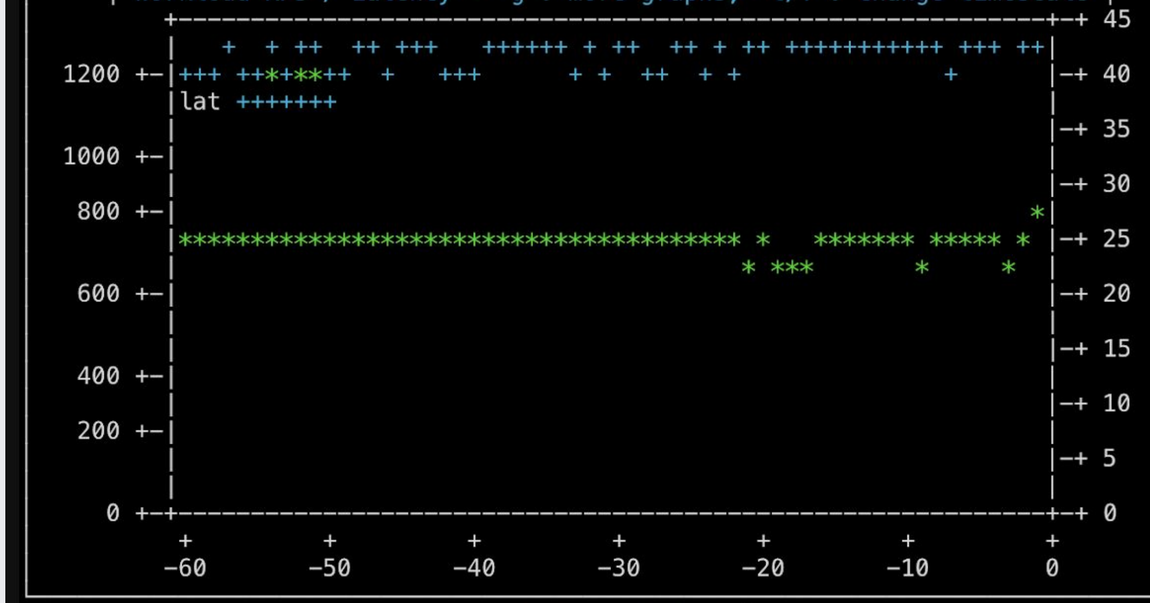
Why the latency deterioration?

- Scheduling inefficiencies
- CPU sub-resource contention. The same instructions take longer to execute as CPU gets saturated.
- Sideloader regulates cpu.max to maintain sufficient CPU headroom.

```
[  Running  ] 2021-01-22 05:01:13 PM
[  config   ] satisfied: 18  missed:  0
[   oomd    ] workload: +pressure -senpai  system: +pressure -senpai
[ sideload  ] jobs:  1/ 1  failed:  0  cfg_warn:  0  -overload -crit
[  sysload  ] jobs:  0/ 0  failed:  0
[ workload  ] load: 59.0%  lat:  44ms  cpu: 39.9%  mem: 24.0G  io:  2.4M
```

	cpu%	mem	swap	rbps	wbps	cpuP%	memP%	ioP%
workload	40.4	24.9G	772K	2.7M	4.3M	5.9	-	0.1
sideload	13.6	3.0G	-	32.0K	32.0M	39.3	34.7	34.7
hostcritical	6.9	791M	32.0K	-	-	4.8	3.9	3.9
system	-	174M	788K	-	-	-	-	-
user	0.0	84.9M	21.0M	-	-	0.0	-	-
-	64.5	29.5G	578M	2.7M	36.3M	43.9	16.1	16.3

Workload RPS / Latency - 'g': more graphs, 't/T': change timescale



Management logs

```
("failed")
[16:58:05 rd-agent] [INFO] resctl: Applying updated slice configurations
[16:58:05 rd-agent] [INFO] hashd: Updating "hashd-A" to lat=100.00ms@95.00%
rps=745 mem=33.84% log=1.05Mbps frac=1.00
[16:58:05 rd-agent] [INFO] svc: "rd-hashd-A.service" started (Running)
[16:58:11 rd-sideloader] OVERLOAD: end, resuming normal operation
[17:00:16 rd-agent] [INFO] side: "compile-job" started
[17:00:16 rd-sideloader] JOB: Starting rd-sideload-compile-job.service
[17:00:16 rd-sideloader] Running as unit: rd-sideload-compile-job.service
```

Other logs

```
[17:01:13 rd-sideload-compile-job]   HDRTEST usr/include/asm-generic/errno.h
[17:01:13 rd-sideload-compile-job]   AR      arch/x86/entry/vsyscall/built-in.a
[17:01:13 rd-sideload-compile-job]   LDS     arch/x86/entry/vdso/vdso.lds
[17:01:14 rd-sideload-compile-job]   AS      arch/x86/entry/entry_64_compat.o
[17:01:14 rd-sideload-compile-job]   CC      arch/x86/entry/vdso/vdso32-setup.o
[17:01:14 rd-sideload-compile-job]   CC      certs/blacklist_nohashes.o
[17:01:14 rd-sideload-compile-job]   CC      arch/x86/realmode/rm/wakemain.o
[17:01:14 rd-sideload-compile-job]   HDRTEST usr/include/asm-generic/param.h
[17:01:14 rd-sideload-compile-job] UNKNOWN
```

[side.sideloader] The Sideloader - 'i': index, 'b': back

Now, let's repeat the experiment from the last section, but launch the Linux compile job as a sideload that's supervised by the sideloader.

rd-hashd should already be running at 60% load. Once it warms up, let's start a Linux compile job with 1x CPU count concurrency as before. It'll have the same resource weights as before; the only difference is that it's now running under the supervision of sideloader:

[Start linux compile job as a sideload]

While the Linux source tree is being untarred, depending on the memory situation and IO performance, you may see a brief spike in latency as the kernel tries to figure out the hot working set of the primary workload. Once the compile jobs start running, sideload starts consuming CPU time and pushing up CPU utilization. Look in the upper right pane, and at the utilization graphs in the graph view ('g'). CPU utilization will go up but won't reach 100%.

Look at the latency graph. It's gone up a bit, but a lot less than before, when we ran it as a sysload. You can change the time scale with 't/T' to compare the latencies before and after. How much sideloads impact latency is determined by the headroom: The bigger the headroom, the lower the CPU utilization, and the lower the latency impact.

The current state, in which the sideload fills up the machine's left-over capacity, with a controlled latency impact on the main workload - can be sustained indefinitely. Let's see how ramping up to full load behaves:

[Set full load]

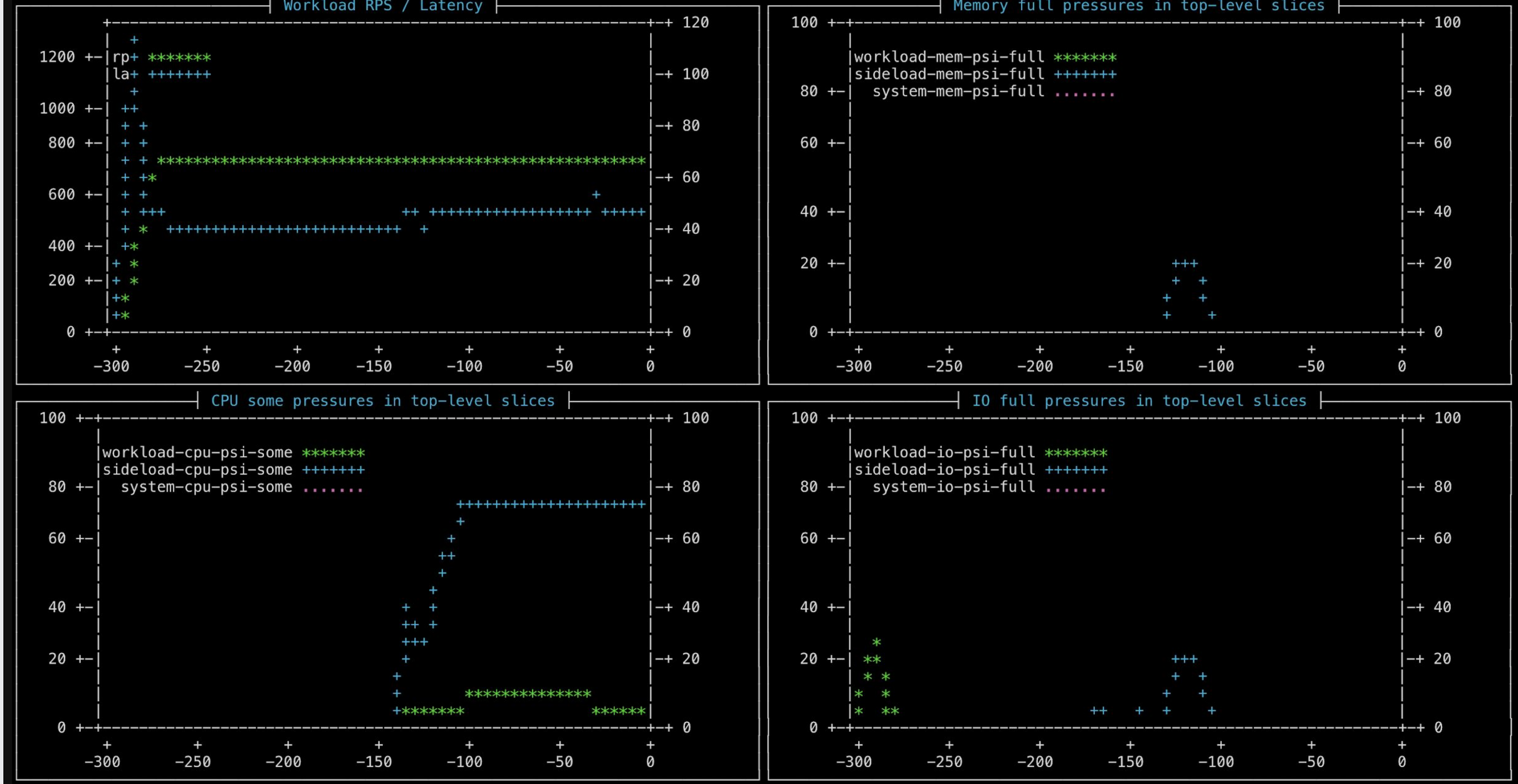
The difference from the no sideload case should be significantly less pronounced. Look at how the sideloader state transitions to overload, freezing the Linux compile job. The build job is configured to expire after being frozen for 30s. Let's wait and see what happens.

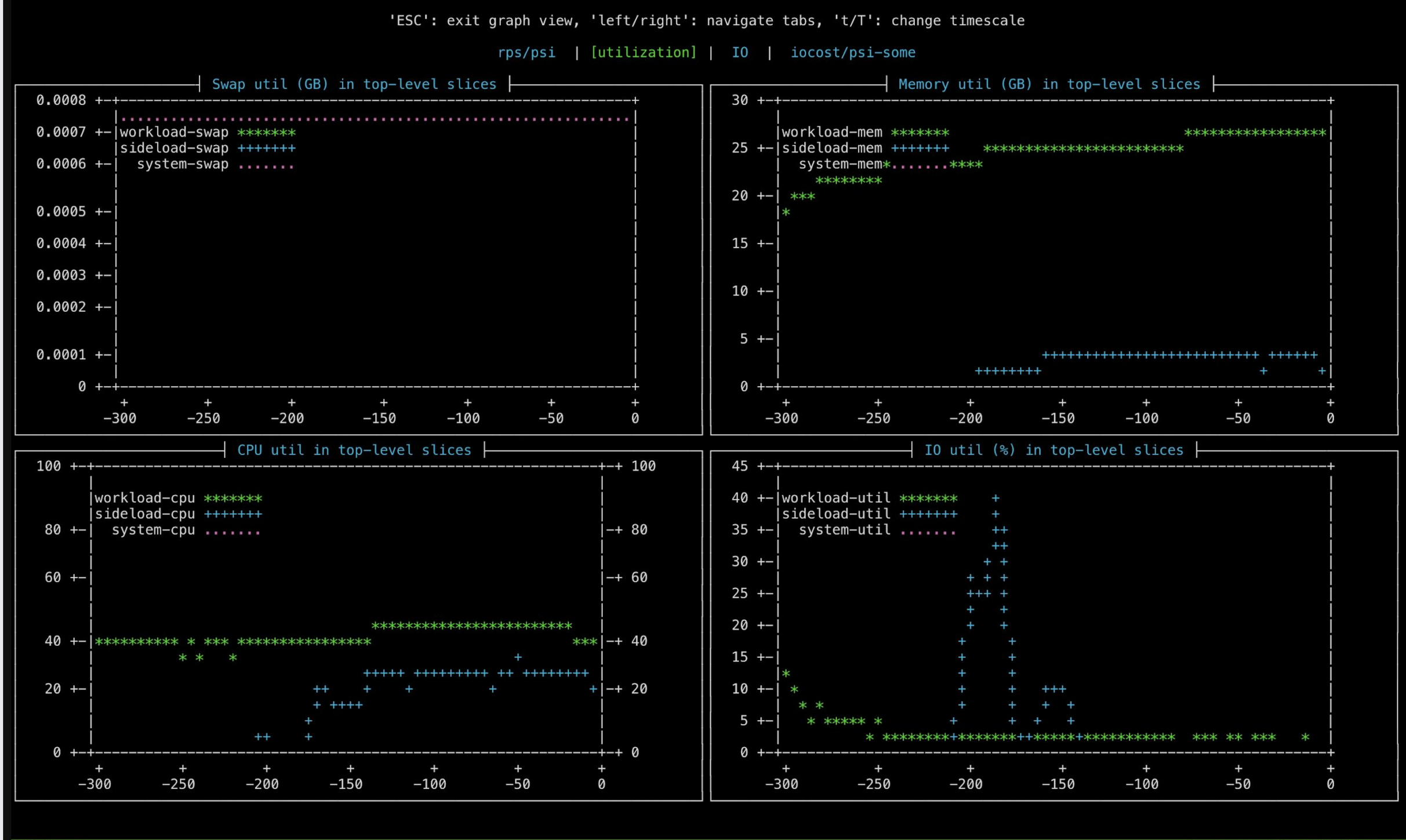
We just demonstrated that, with the help of sideloader, sideloads can utilize the left-over capacity of the machine, with only a controlled and configurable latency impact on the main workload, and without significantly impacting the main workload's ability to ramp up when needed.

Read on

'ESC': exit graph view, 'left/right': navigate tabs, 't/T': change timescale

[rps/psi] | utilization | IO | iocost/psi-some




'ESC': exit graph view, 'left/right': navigate tabs, 't/T': change timescale
rps/psi | [utilization] | IO | iocost/psi-some
Swap util (GB) in top-level slices
workload-swap
sideload-swap
system-swap
Memory util (GB) in top-level slices
workload-mem
sideload-mem
system-mem
CPU util in top-level slices
workload-cpu
sideload-cpu
system-cpu
IO util (%) in top-level slices
workload-util
sideload-util
system-util

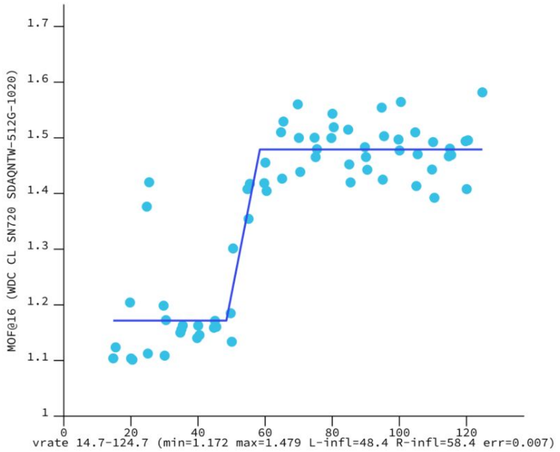
resctl-bench

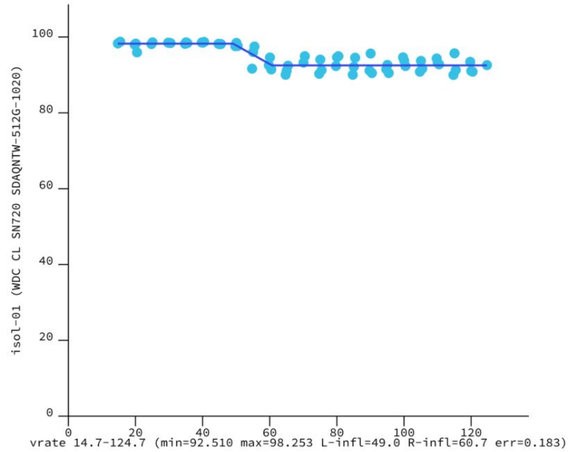
Why?

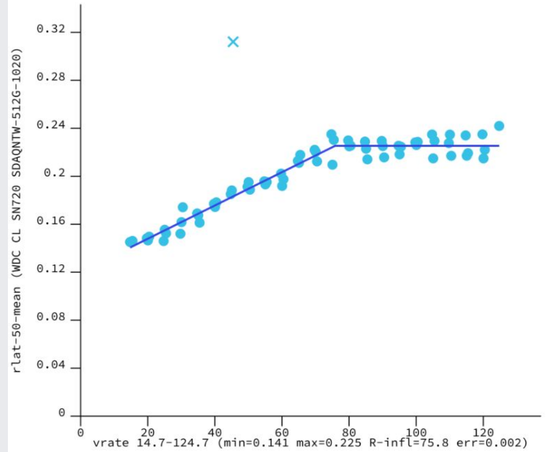
- System behavior under resource contention too complex to capture in simple benchmarks.
- Simulate the entire system operations instead.

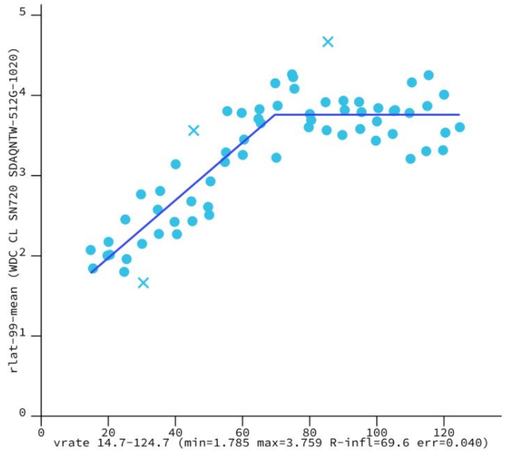
e.g.: iocost-tune benchmark

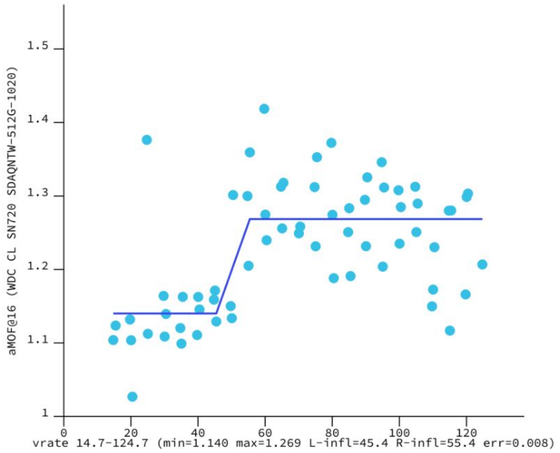
- io.cost IO controller takes a dozen configuration parameters. Tuning QoS params is challenging.
- Simulate the full isolation scenarios and try out different QoS levels to understand overall behavior.
- Calculate solutions based on the identified behavior trends.

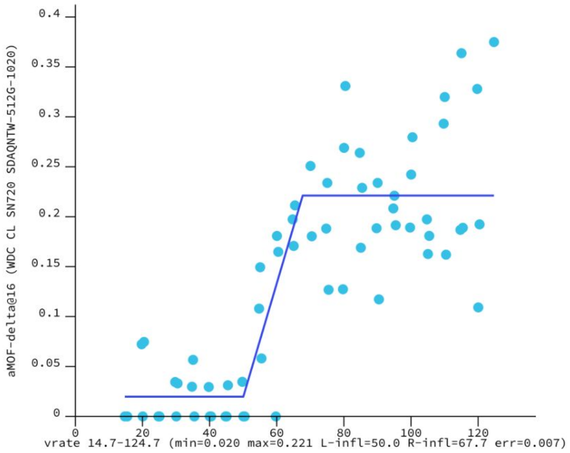

MOF@16 (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=1.172 max=1.479 L-infl=48.4 R-infl=58.4 err=0.007)

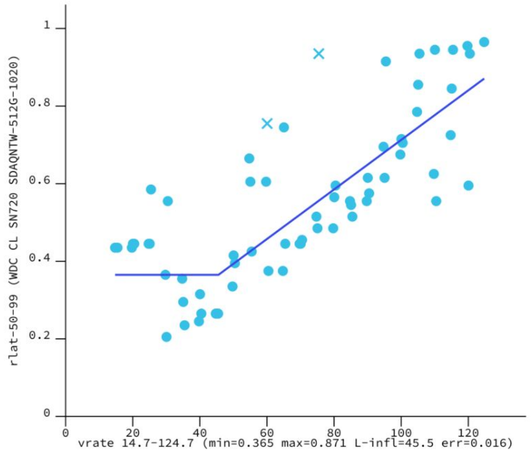

isol-01 (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=92.510 max=98.253 L-infl=49.0 R-infl=60.7 err=0.183)

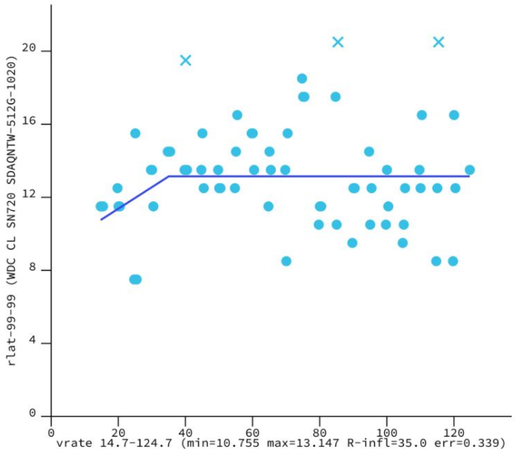

rlat-50-mean (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=0.141 max=0.225 R-infl=75.8 err=0.002)

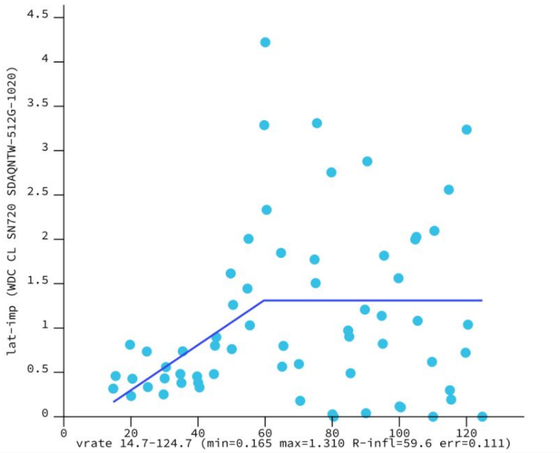

rlat-99-mean (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=1.785 max=3.759 R-infl=69.6 err=0.040)

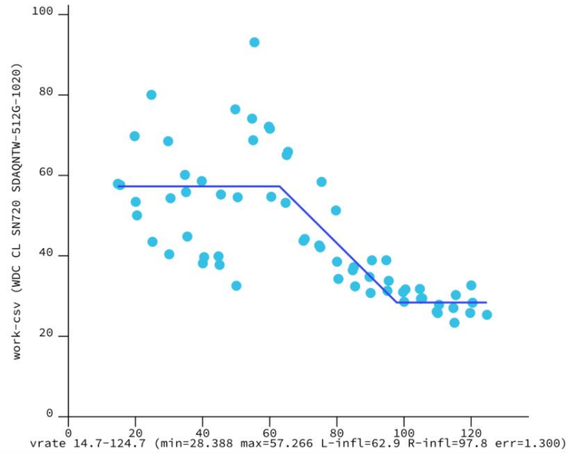

aMOF@16 (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=1.140 max=1.269 L-infl=45.4 R-infl=55.4 err=0.008)

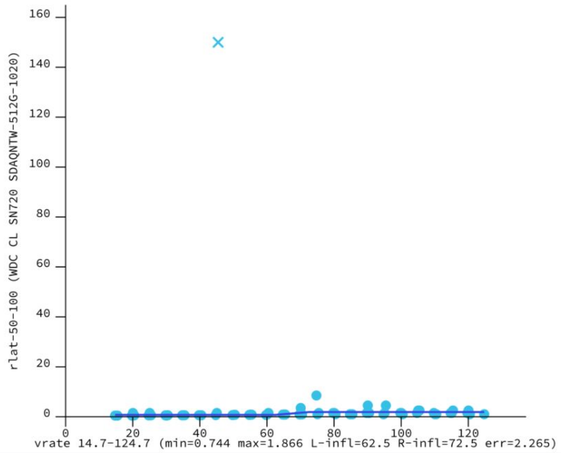

aMOF-delta@16 (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=0.020 max=0.221 L-infl=50.0 R-infl=67.7 err=0.007)

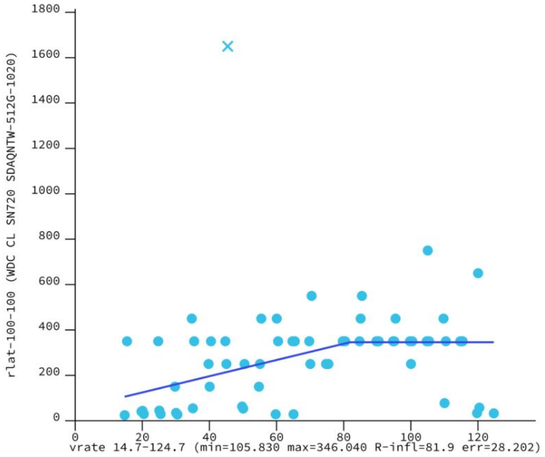

rlat-50-99 (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=0.365 max=0.871 L-infl=45.5 err=0.016)


rlat-99-99 (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=10.755 max=13.147 R-infl=35.0 err=0.339)


lat-imp (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=0.165 max=1.310 R-infl=59.6 err=0.111)


work-csv (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=28.388 max=57.266 L-infl=62.9 R-infl=97.8 err=1.300)


rlat-50-100 (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=0.744 max=1.866 L-infl=62.5 R-infl=72.5 err=2.265)


rlat-100-100 (WDC CL SN720 SDAQNTW-512G-1020)
vrate 14.7-124.7 (min=105.830 max=346.040 R-infl=81.9 err=28.202)

iocost-tune Solutions

```
...

[bandwidth] MOF=max
  info: scale=64.28% MOF=1.479@16 aMOF=1.269 aMOF-delta=0.182 isol-01=92.51%
  rlat: 50-mean= 209u 50-99= 485u 50-100= 940u 99-mean= 3.6m 99-99=13.1m 100-100= 283m
  wlat: 50-mean=54.9u 50-99= 305u 50-100= 2.6m 99-mean= 863u 99-99=12.2m 100-100= 378m
  model: rbps=1367869845 rseqiops=147417 rrandiops=143285 wbps=638142628 wseqiops=136834 wrandiops=58509
  qos: rpct=0.00 rlat=3566 wpct=0.00 wlat=863 min=100.00 max=100.00

[isolation] isolation (aMOF-delta=min)
  info: scale=45.04% MOF=1.172@16 aMOF=1.140 aMOF-delta=0.020 isol-01=98.25%
  rlat: 50-mean= 183u 50-99= 365u 50-100= 744u 99-mean= 2.9m 99-99=13.1m 100-100= 214m
  wlat: 50-mean=54.9u 50-99= 305u 50-100= 2.6m 99-mean= 584u 99-99= 6.6m 100-100= 378m
  model: rbps=958300453 rseqiops=103277 rrandiops=100383 wbps=447069121 wseqiops=95863 wrandiops=40990
  qos: rpct=0.00 rlat=2874 wpct=0.00 wlat=584 min=100.00 max=100.00

...
```

https://facebookmicrosites.github.io/resctl-demo-website

facebook