Latency_nice

Implementation and Use-case for Scheduler Optimization

Parth Shah <parth@linux.ibm.com> IBM
Chris Hyser <chris.hyser@oracle.com> Oracle
Dietmar Eggemann <dietmar.eggemann@arm.com> Arm

Agenda

- *Design & Implementation*
  - per-task
  - Privileges
  - per-cgroup
- *Use-cases*
  - Scalability in Scheduler Idle CPU Search Path
  - EAS
  - TurboSched - task packing latency nice > 0 tasks
  - Idle gating in presence of latency-nice < 0 tasks

Design & Implementation

1) per-task
2) privileges
3) per-cgroup

per-task

- What it describes?
  - Analogues to task NICE value but for ***latency hints***
  - Per-task attribute (**syscall**, cgroup, etc. Interface may be used)
  - A r**elative value :**
    - Range = **[-20, 19]**
    - Low latency requirements = higher value compared to other tasks
    - value = -20 : task is latency sensitive
    - Value = 19 : task does not care for latency at all
    - Default value = 0
- Proposed Interface in review: **sched_setattr()** existing syscall

https://lkml.org/lkml/2019/9/30/215

Privileges

- Can non-root user **decrease value** of latency_nice?
  - i.e., can the task be promoted to have indicate lower latency requirements?
- Use ***CAP_SYS_NICE*** capability to restrict non-root user lower the value. This makes it **analogues to task NICE.**
- Pros:
  - Only System Admin can promote the task
  - A task once demoted by Admin, user no longer can promote it. Mitigates DoS attacks.
- Cons:
  - A user cannot lower the value of owned tasks.
  - A user once increased the value cannot set its value to the default 0.
- Use-cases already in discussion:
  - Reduce core-scan search for latency sensitive tasks
  - Pack latency tolerant tasks on fewer CPUs to save energy (EAS/TurboSched)
- Ideas in the community:
  - Be conservative: Introduce this capability based on the use-case introduced
  - Currently, none of the proposed use-case allows DoS like attacks

https://lkml.org/lkml/2020/2/28/

per-cgroup - why?

- **prefer-idle** feature – bias CPU selection towards the least busy one to improve wakeup latency
- Pixel4 (v4.14 based)

  none on /dev/stune type cgroup (rw,nosuid,nodev,noexec,relatime,schedtune)

  flame:/ # find /dev/stune/ -name schedtune.prefer_idle

  ./**foreground**/schedtune.prefer_idle **1**

  ./**rt**/schedtune.prefer_idle 0

  ./**camera-daemon**/schedtune.prefer_idle **1**

  ./**top-app**/schedtune.prefer_idle **1**

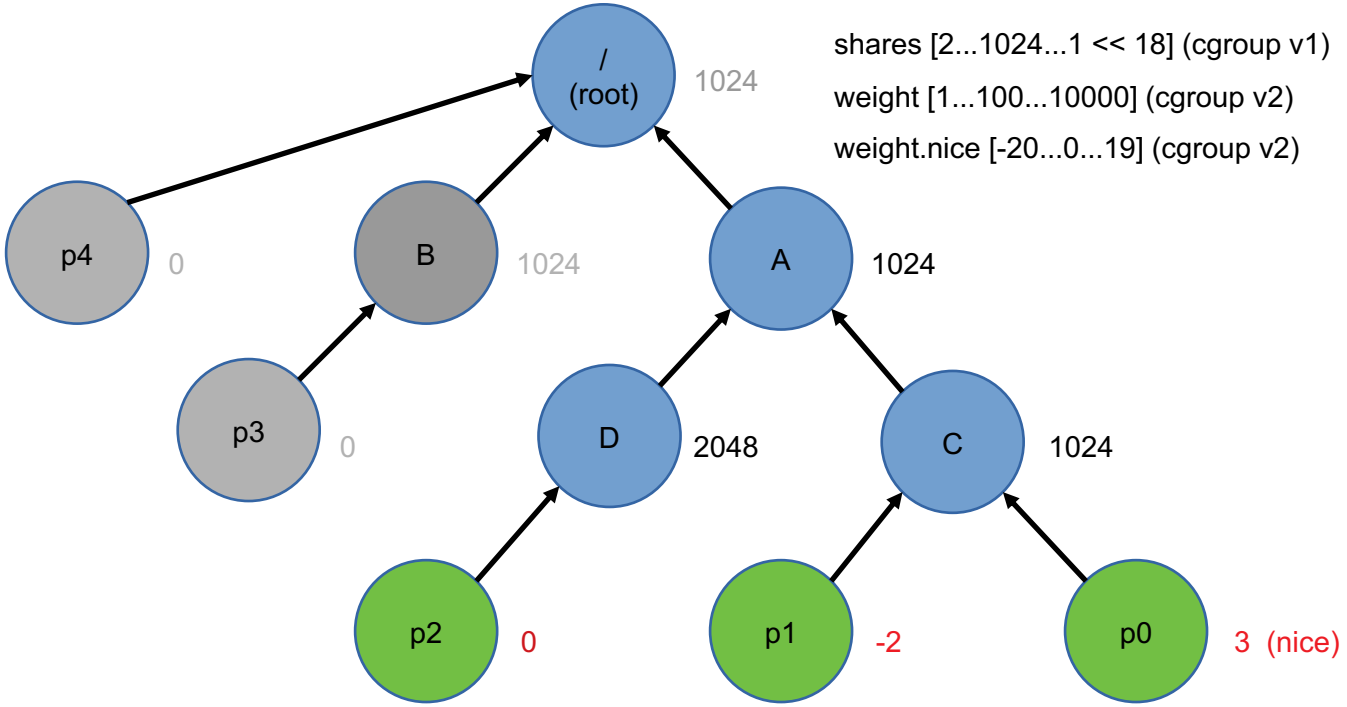
  ./**background**/schedtune.prefer_idle 0

per-cgroup - definition

- cgroup
  - mechanism to organize processes hierarchically & distribute system resources along the hierarchy
  - resource distribution models:
    - weight: [1, **100**, 10000], symmetric multiplicative biases in both directions
    - limit: [0, **max**], child can only consume up to the configured amount of the resource
    - protection: [**0**, max], cgroup is protected up to the configured amount of the resource
- CPU controller
  - regulates distribution of CPU cycles (time, bandwidth) as system resource
  - absolute bandwidth limit for CFS and absolute bandwidth allocation for RT
  - utilization clamping (boosting/capping) to e.g. hint schedutil about desired min/max frequency

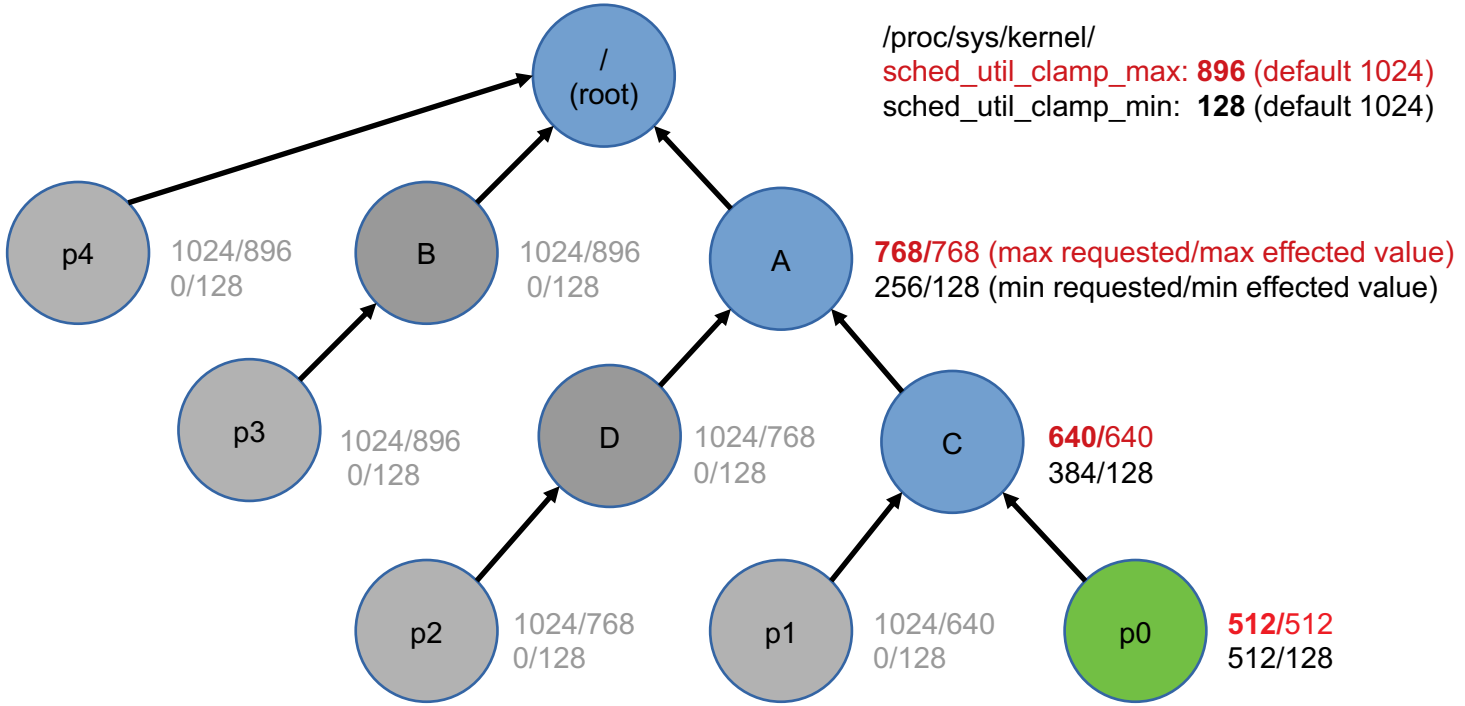
per-cgroup - cpu controller - nice & shares

- sched_**prio**_to_**weight**[40] = { 88761 (-20), ... 1024 (0), ... 15 (19) }
- nice to weight: weight = 1024/(1.25)^(nice)
- relative values affect the proportion of CPU time (weight)



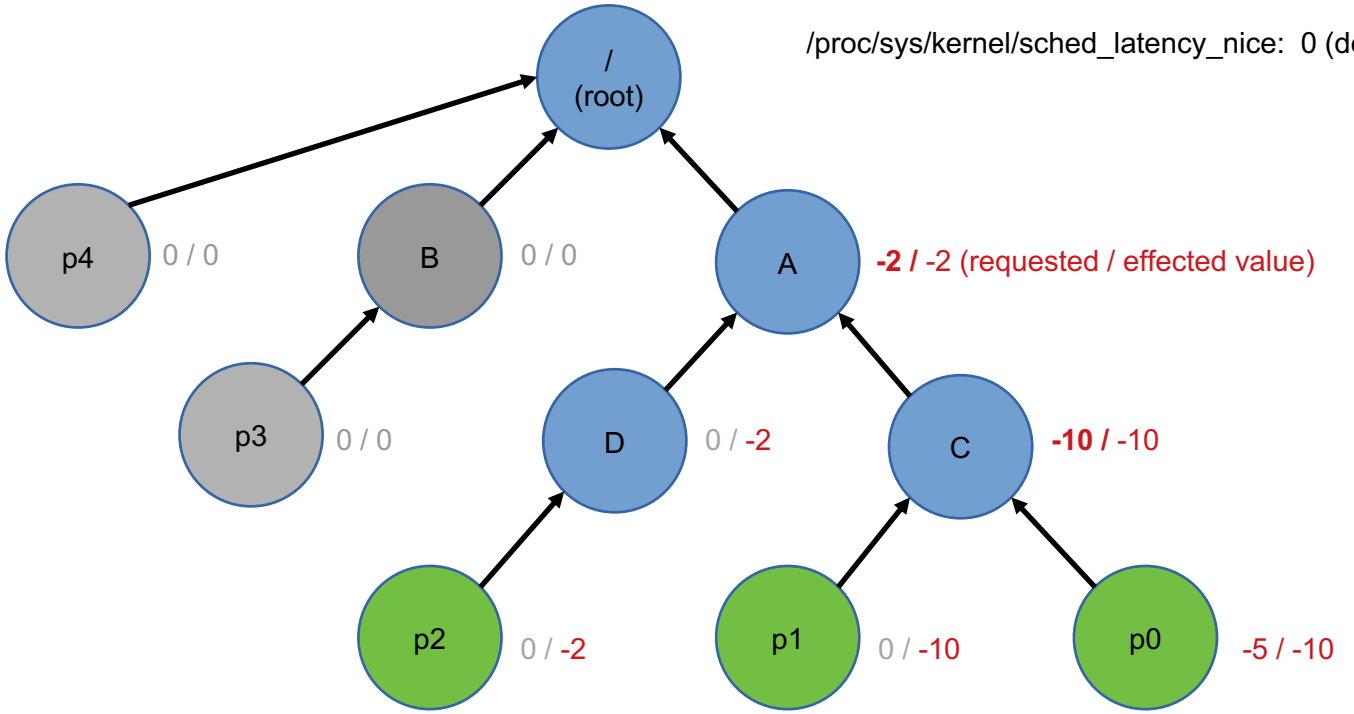
per-cgroup - cpu controller - uclamp.min/max

- task effective value restricted by task (user req), cgroup hierarchy & system-wide setting
- clamping is boosting (protection) via uclamp.min & capping (limit) via uclamp.max



per-cgroup - cpu controller - latency nice ?

- system resource has to be CPU cycles
- resource distribution model: limit would work for negative latency_nice values [-20, 0]
- update (aggregation) – where ?



Use cases

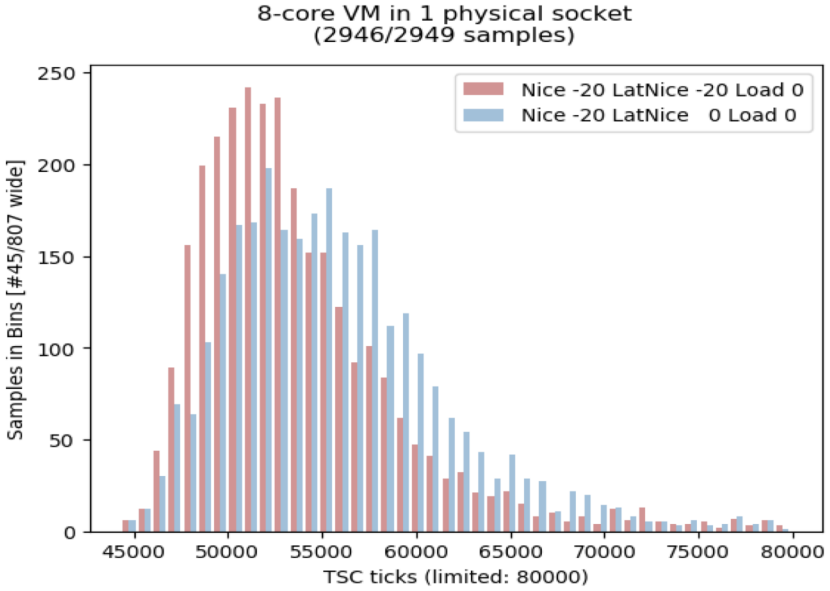
1) Scheduler Scalability (ORACLE)
2) EAS (Android)
3) TurboSched (IBM)
4) IDLE gating (IBM)

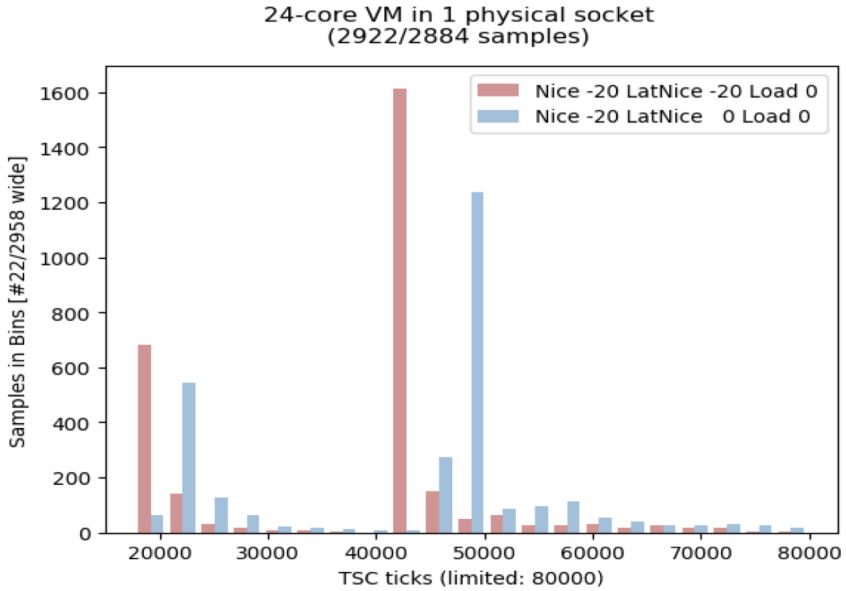
Scalability in Scheduler Idle CPU Search Path

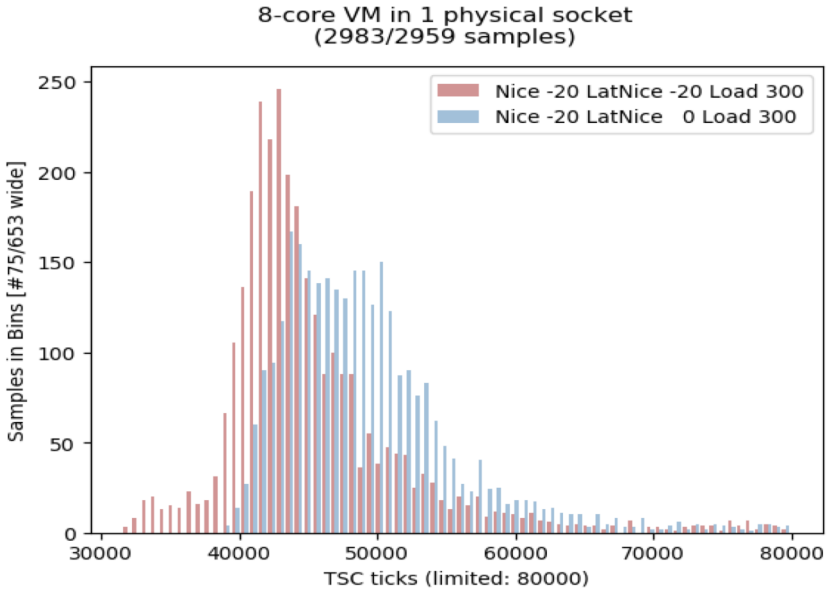
- Patchset author identified CFS 2nd-level scheduling domain idle cpu search as source of wakeup latency
  - **Skipping search for certain processes improved TPCC by 1%.**
  - Certain critical communication processes are very short-lived and sensitive to latencies
  - Real-time avoided because of interop issues with cgroups
- Start by understanding scope of problem
  - Some number of 100% cpu bound load processes to fill queues
  - Target process (running at desired latency nice value) and measuring process
    - Target does eventfd_read()
    - Measurer grabs TSC, does eventfd_write()
    - Target wakes and grabs TSC (in same socket)
    - Target communicates value back to measurer

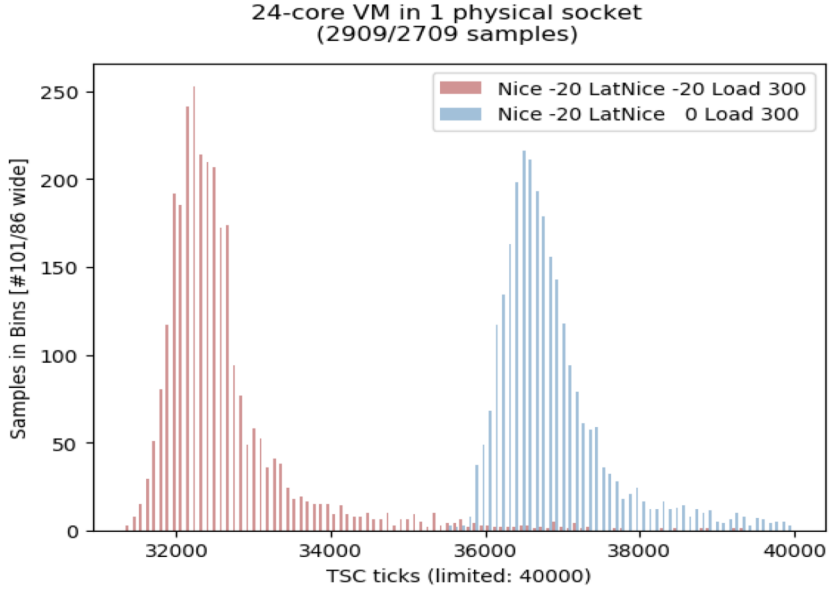
Is Skipping the Search Visible?

(Early Experiment Numbers)









EAS

- **prefer_idle** replacement
- avoid latency from using Energy Model (EM) for certain taskgroups
- look for idle CPUs/best fit CPU for **latency_nice = -1** tasks instead
- latency_nice = [-1, 0] [don't use EM, use EM]

- Testcase
  - test UI performance with Jankbench
    - measure number of dropped or delayed frames (jank)

TurboSched: Task packing

- Discussed at OSPM-III
- Pack ***small background tasks*** on fewer cores.
- This reduces power consumption => allows busy core to sustain/boost Turbo frequencies for longer durations.
- ***small background tasks:***
  - *P->latency_nice > 15 && task_util(p) < 12.5%*
- ***Result***
  - Spawn 8 important tasks
  - Spawn 8 small noisy tasks waking up randomly doing while(1) with latency_nice= 19
  - Noisy tasks get packed on busier cores, channeling power to other cores by maintaining power budget
  - This boosts busier cores to higher frequency
  - Seen upto 14% performance benefit in throughput for important tasks

https://lkml.org/lkml/2020/1/21/39

IDLE gating in presence of latency-nice<0 tasks

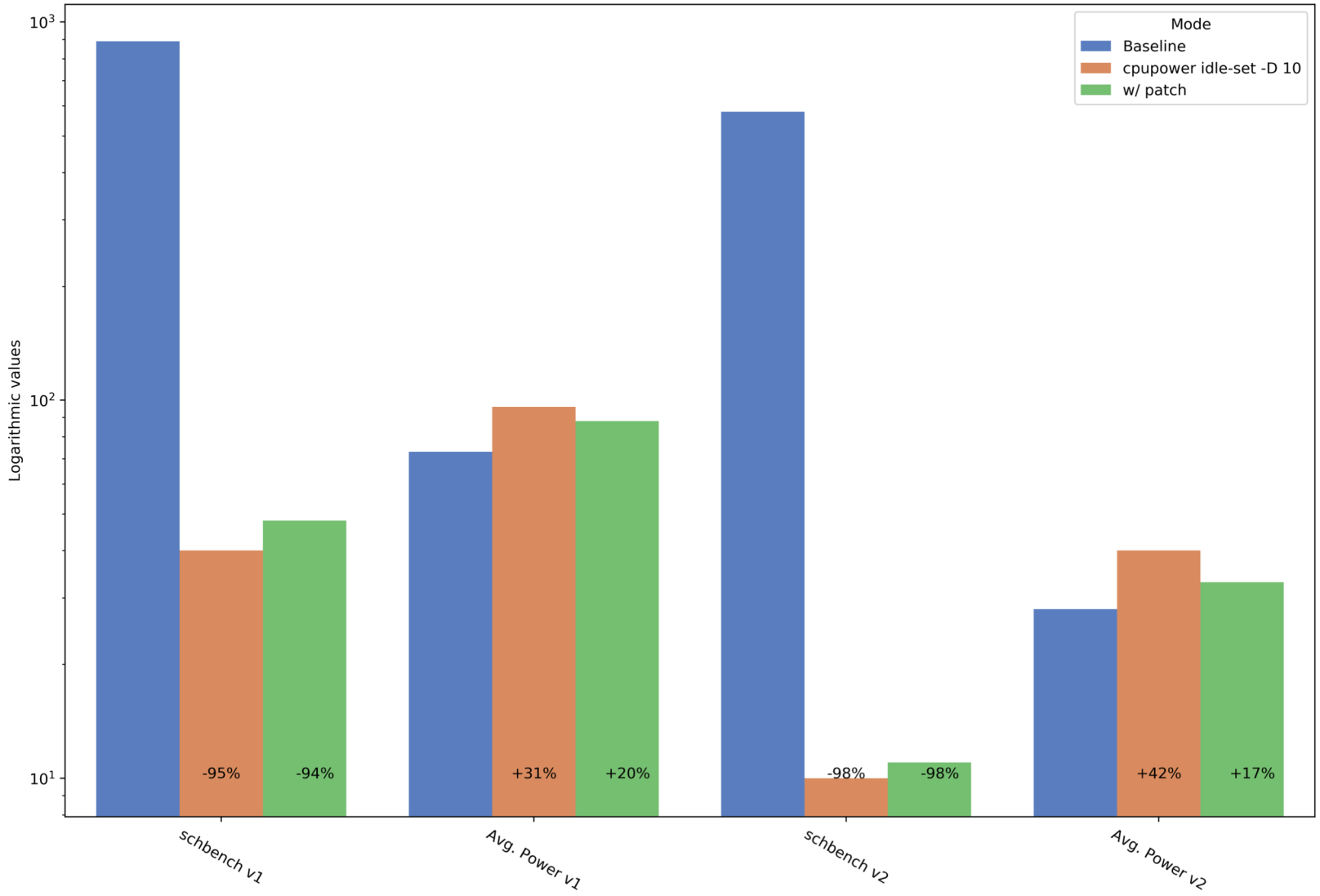
- PM_QoS:
  - Restrict IDLE states based on exit_latency
  - Its per-device or system-wide configuration
  - No per-task control mechanism
  - Problem gets intense with multi-core multi-thread systems
- Latency_nice can hint CPUs on latency sensitive tasks
- Implementation:
  - per-cpu counter to track latency sensitive tasks
  - Increase/decrease this counter upon task entering/exiting scheduler domain
  - Restrict the call to CPUIDLE governor if any latency-sensitive tasks exists
- Benefits:
  - Only the **CPU executing latency_sensitive marked tasks won't go idle**
  - **Other CPUs still goes to IDLE states** based on CPUIDLE governor decision
  - Best for performance, by cutting IDLE states latency
  - Better than disabling all IDLE states
  - Allows Turbo frequency to boost by saving power on IDLE CPUs

https://lkml.org/lkml/2020/5/7/577

Results: schbench

Benchmarks:

- v1:
  - schbench –r 30
- v2:
  - schbench –m 2 –t 1



% values are w.r.t. Baseline

Results: pgbench

44 Clients running in parallel
$> pgbench –T 30 –S –n –R 10 –c 44

	Baseline	cpupower idle-set –D 10	w/ patch
Latency avg. (ms)	2.028	0.424 (-80%)	1.202 (-40%)
Latency stddev	3.149	0.473	0.234
Trans. completed	294	304 (+3%)	300 (+2%)
Avg. Energy (Watts)	23.6	42.5 (+80%)	26.5 (+20%)

1 Client running
$> pgbench –T 30 –S –n –R 10 –c 1

	Baseline	cpupower idle-set –D 10	w/ patch
Latency avg. (ms)	1.292	0.282 (-78%)	0.237 (-81%)
Latency stddev	0.572	0.126	0.116
Trans. completed	294	268 (-8%)	315 (+7%)
Avg. Energy (Watts)	9.8	29.6 (+30.2%)	27.7 (+282%)

% values are w.r.t. Baseline

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of the employers (IBM Corporation).
- IBM and IBM (Logo) are trademarks or registered trademarks of International Business Machines in United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product and service names may be trademarks or service marks of others.

References

1. Introduce per-task latency_nice for scheduler hints, https://lkml.org/lkml/2020/2/28/166
2. Usecases for the per-task latency-nice attribute, https://lkml.org/lkml/2019/9/30/215
3. TurboSched RFC v6, https://lkml.org/lkml/2020/1/21/39
4. Task latency-nice, https://lkml.org/lkml/2020/1/21/39
5. IDLE gating in presence of latency-sensitive tasks, https://lkml.org/lkml/2020/5/7/577
6. ChromeOS usecase, https://lkml.org/lkml/2020/4/20/1353