The "Thing" that was "Latency Nice"

Let's review the Use-Cases we have and find out what should be the best API
Introduction

- **The question:** how to **dynamically tune** the task **wakeup path** for certain classes of **workloads** and usage scenarios?
- **The problem:** different use-cases have different **contrasting needs**
  - reduce wakeup latency (e.g. by looking at fewer CPUs or preempting current)
  - find a better wakeup CPU (e.g. by looking at more CPUs or finding an "optimal" one)
- **The story so far:** at **OSPM** we had a **discussion** "trying" to fit different needs into a single knob (latency_nice)

- **Lesson learnt:** we need to put more effort on **defining the requirements**
  - we got a template[^1] meant to **collect requirements** and (possibly) **surface commonalities**

[^1]: https://lore.kernel.org/lkml/87imfi2qbk.derkling@matbug.net/
## Use-cases Requirements

<table>
<thead>
<tr>
<th></th>
<th>skip idle cpus</th>
<th>skip energy_aware</th>
<th>tasks packing</th>
<th>vruntime bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target behavior</strong></td>
<td>Idle cpu search during wakeup <em>trades throughput for latency</em></td>
<td>EAS cpu search during wakeup <em>trades energy saving for latency</em></td>
<td>Wakeup prefers idle core is energy inefficient for latency tolerant tasks</td>
<td>Latency tolerant tasks do not preempt</td>
</tr>
<tr>
<td><strong>Desired behavior</strong></td>
<td>Skip some/all cpu searched for LS tasks</td>
<td>Skip EAS wake-up path for LS tasks, fallback into sis()</td>
<td>Add a new wake-up path for LT task to select a busy core beyond LLC</td>
<td>Tune the &quot;vruntime bonus&quot;, higher for LS, smaller for LT</td>
</tr>
<tr>
<td><strong>Existing knobs</strong></td>
<td>N/A</td>
<td>None in mainline, “Prefer idle” in Android</td>
<td>N/A</td>
<td>0.5*sysctl_sched_latency (hard-coded for all tasks)</td>
</tr>
<tr>
<td><strong>Proportionality</strong></td>
<td>Specify num idle cpus in sched domain to search</td>
<td>N/A</td>
<td>N/A</td>
<td>Could be added to vdiff, <code>vdiff &gt; wakeup_gran(se)</code></td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>[0..min(size(SD), ALL)]</td>
<td>{0,1}</td>
<td>{0,1}</td>
<td>[0..sysctl_sched_latency]</td>
</tr>
<tr>
<td><strong>Desired APIs</strong></td>
<td>PT</td>
<td>PT, TG</td>
<td>PT, TG</td>
<td>SW, PT, TG</td>
</tr>
<tr>
<td><strong>Mapping Example</strong></td>
<td>Mapping from [-20,19] [-20,-1]: search 20 + n [0,19]: search ALL</td>
<td>Binary mapping from range e.g. [-20,-1]:1 [0,19]:0</td>
<td>Binary mapping from range e.g. [-20,0]:0 [1,19]:1</td>
<td>Linear mapping from range [-20, 19]:[sched_latency:0]</td>
</tr>
</tbody>
</table>

**LT** = Latency-Tolerant, **LS** = Latency-Sensitive, **SW** = System-Wide, **PT** = Per-Task and **TG** = per Task-Groups

[1] [https://lore.kernel.org/lkml/87/imfi2qbk.derkling@matbug.net/](https://lore.kernel.org/lkml/87/imfi2qbk.derkling@matbug.net/)
Discussion Points

Here we are at reviewing and comparing the collected requirements and addressing these main questions:

1. Which of the different use-cases can **work together**?

2. Do we have a case for search **less -vs- more** CPUs?

3. What about **task group** support?
   - which use-cases can benefit from?

4. Does it makes sense to use a **unified API**?
   - does it help to enforce consistency among co-existing use-cases?
   - if it’s not being called ‘nice’, should we use a different range or set of values/flags?

5. What about a **use-case dedicated** set of **per-task attributes**?
   - should be via `sched_setattr()`?