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[1]

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

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

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

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()`?