Localising the system latency/throughput/power tunability surface

Patrick Bellasi (ARM)
SchedTune Main Goals and Discussions Points

● Enable the collection of task related information from informed runtimes
  ○ do we want to support tasks classification and per-task tuning of scheduler behaviors?

● OPP Selection: running tasks at higher/lower OPP
  ○ is it acceptable to bias how schedutil selects the frequency?
  ○ should we do that depending on which tasks are currently RUNNABLE on that CPU?

● Task Placement: biasing CPU selection in the wake-up path
  ○ is it acceptable to bias where the CFS scheduler place a task?
  ○ can we force tasks on more/less capable CPUs independently from their utilization?

● Use of CGroups to collect tasks related information
  ○ is that an acceptable interface?
  ○ should we use a dedicated new controller (e.g. [1]) or extend an existing one?

● Validation of the expected behaviors
  ○ can we define a set of (synthetic) use-cases and expected behaviors?

# SchedTune New Design Proposal

<table>
<thead>
<tr>
<th><strong>SchedTune</strong></th>
<th><strong>Extending CPU Contoller</strong></th>
</tr>
</thead>
</table>
| Boost value            | Using the existing `cpu.shares` attribute.  
- by default tasks have a 1024 share  
- boosted tasks gets a share >1024 (more CPU time to run)  
- negative boosted tasks gets <1024 (less CPU time to run) |
| OPP biasing            | Add a new `cpu.min_capacity` attribute. Tasks in the group are *granted to be scheduled* on a CPU which provides at *least the required minimum capacity* |
| Negative boosting      | Add a new `cpu.max_capacity` attribute. Tasks in the group are *never scheduler* (when alone) on a cpu with CPU capacity higher that this value. |
| CPU selection and prefer_idle | The `cpu.shares` value can be used as a “flag” to know when a task is boosted. E.g. is cpu.shares > 1024 (or another configurable threshold value) we look for an idle CPU.  
The `cpu.[min|max]_capacity` can also bias the selection of a big|LITTLE CPU. |
| Latencies reduction    | Tasks with higher `cpu.shares` value are entitled more CPU time and this turns out to give them better chances to get scheduled by preempting other tasks with lower shares.  
**NOTE**: the CPU bandwidth not consumed by high `cpu.shares` value tasks is still available for tasks with lower shares. |
Backup slides

Current SchedTune Concepts and Implementation Details
Performance Boosting: What Does it Mean?

- Speedup the time-to-completion for a task activation
  - by running at an higher capacity CPU (i.e. OPP)
    - i.e. small tasks on big cores and/or using higher OPPs
- To achieve such a goal we need:
  - A) Boosting strategy
    - Evaluate how much “CPU bandwidth” is required by a task
  - B) CPU selection biasing mechanism
    - Select a Cluster/CPU which (can) provide that bandwidth
    - Evaluate if the energy-performance trade-off is acceptable
  - C) OPP selection biasing mechanism
    - Configure selected CPU to provide (at least) that bandwidth
    - ... but possibly only while a boosted task is RUNNABLE on that CPU
  - ... do all that with no noticeable overhead
Patches Availability and List Discussions

- The initial full stack has been split in two series
  - 1) Non EAS dependant bits
    - OPP selection biasing
    - Global boosting strategy
    - CGroups based per-task boosting support
  
  - 2) EAS dependant bits
    - CPU selection biasing
    - Energy model filtering

Posted on LKML as RFCv1[1] and RFCv2[2]

Available on AOSP and LSK for kernels 3.18 and v4.4 [3,4]
Boosting Strategy: Bandwidth Margin Computation

- Task utilization defines the task's required CPU bandwidth
  - To boost a task we need to inflate this requirement by adding a “margin”
  - Many different strategies/policies can be defined

- Main goals
  - Well defined meaning from user-space
    - 0% boost run @ min required capacity (MAX energy efficiency)
    - 100% boost run @ MAX possible speed (min time to completion)
    - 50%? ==> “something” exactly in between the previous two
  - Easy integration with SchedFreq and EAS
    - By working on top of already used signals
    - Thus providing a different “view” on the SEs/RQs utilization signals
Signal Proportional Compensation (SPC)

- The boost value is converted into an additional margin
  - Which is computed to compensate for max performance
    - i.e. the boost margin is a function of the current and max utilization

\[ \text{margin} = \text{boost pct} \times (\text{max capacity} - \text{cur capacity}) \], \text{boost pct} \in [0,1]

Ramp task: 5-60% @5% steps every 3[s] – SPC boost @30%
OPP Selection Biasing Mechanism

- Goal: account for boost margin on OPP selection
- Use RQ's boosted_utilization defined using:
  - Global boost value, when using global boosting
  - MAX boost-group's boost value, when using per-task boosting
CPU Selection Biasing Mechanism (1/3)

- Energy-Aware Wakeup Path
  Goal: find a CPU which can host the boosted utilization
  ○ using the boosted_utilization signal on some EA wakeup checks

![Diagram of CPU selection process]

Example of CPU selection for a 10% task with a 50% boost:

- Cluster Selection
  - cluster sd
  - CPU Selection
  - if (next_cpu != prev_cpu)

Cluster MAX capacity \( \geq \) boosted_utilization

CPU (curr/next) capacity \( \geq \) boosted_utilization

![Bar chart showing CPU utilization and capacity levels]
CPU Selection Biasing Mechanism (2/3)

- Evaluation of the Energy-Performance trade-off
  Goal: evaluate if the **increased energy consumption** is compensated by a “reasonable” **performance gain**

- Running small tasks on higher capacity CPUs requires more power
- Performance boost is computed by the EM evaluation step

How much power are we willing to spend to get a certain speedup on time-to-completion?
CPU Selection Biasing Mechanism (3/3)

- PE Space Filtering
- 4 Performance-Energy Space Regions
- 2 'cuts', mapped to the same boost knob value
- “Standard” EAS behaviour for boost=0
  ○ I.e. vertical cut
SchedTune OPP Boosting

RTApp Generated RAMP Task

```
"r5_10-60" : {
  "kind"   : "Ramp",
  "params" : {
    "period_ms" : 16,
    "start_pct" : 5,
    "end_pct"   : 60,
    "delta_pct" : 5,
    "time_s"    : 1,
    "cpus"      : [7],
  },
  "tasks" : 1,
},
```

Clusters Frequencies

Ramp_5_10_60 no boosting

Clusters Frequencies

Ramp_5_10_60 boost @30%

CPU Capacity Biasing
CPU Frequency Selection

- The higher the boost value the higher the avg frequency in this example the task is pinned to run on LITTLE

Ramp task: 5-60% @5% steps every 3[s]
Performance Evaluation (1/2)

- RT-App extended to report slack time related metrics
  - too pessimistic on single period missing
    - keep adding negative slack even if the following activations complete in time
    - can be solved by resetting the metrics at each new activation
- Linaro proposed a “dropped-frames” counter
  - we should integrate that as well

\[
\begin{align*}
\text{MaxSlack} &= \text{Period}_{\text{conf}} - \text{RunTime}_{\text{conf}} \\
\text{PerfIndex} &= \frac{\text{Period}_{\text{conf}} - \text{RunTime}_{\text{meas}}}{\text{MaxSlack}} \\
\text{NegSlack}_{\text{percent}} &= \frac{\sum \text{Max}(0, \text{RunTime}_{\text{meas}} - \text{Period}_{\text{conf}})}{\sum \text{RunTime}_{\text{meas}}}
\end{align*}
\]
Performance Evaluation (2/2)

- Slack Time Distribution

No boosting

- ~61%

SPC 45% boost

- ~93%

Ramp task: 5-60% @5% steps every 3[s]
SchedTune Performance Index

- Based on the composition of two metrics

\[
\text{Perf}_{\text{idx}} = \text{SpeedUp}_{\text{idx}} - \text{Delay}_{\text{idx}}
\]

- SpeedUp_{\text{idx}}: how much faster can the task run?

\[
\text{SpeedUp}_{\text{idx}} = \text{SUI} = \text{cpu\_boosted\_capacity} - \text{task\_util}
\]

- Delay_{\text{idx}}: how much slowed-down can the task be?

\[
\text{Delay}_{\text{idx}} = \text{DLI} = 1024 \times \text{cpu\_util} / (\text{task\_util} + \text{cpu\_util})
\]