Measuring performance efficiency of Linux frequency scaling

Author: Luca Ceragioli

Mailto: l.ceragioli [at] sssup [dot] it

Date: July 21 2021

Repository: https://github.com/VirtuContraFurore/Linux_governor_efficiency

Introduction

This project aims to measure how different cpu governors behave in meeting the deadlines to serve awaiting tasks. CPU frequency scaling is a key technique in mobile power constrained devices. Since modern CPU cores support different clock frequencies, it is possible to reduce the power consumption of the CPU by lowering its operating frequency while the CPU load is below a certain threshold.

In the Linux Kernel, the software component responsible for setting the correct frequency of the CPU cores given the current system load is called **governor**. The governor has key role to determine if the system would be more inclined in saving power rather than using always its maximum performance.

Governor operation theory

The main infrastructure for cpu frequency scaling is named "**cpufreq**" and it provides an Hardware Abstraction Layer (HAL) to read and set the desired cpu frequency for each core. Since each hardware has its own rules there is the risk of having more constraints. For istance, some SoCs only allow one frequency value to be shared among each cores, while others could group the cores which shares the same frequency. There are nonetheless some physical constraints as well, like the CPU temperatures, which must not exceed certain device specific threesholds. Depending on the system, usually two threesholds are defined, also called "**trip points**" or "**operating performance point**", which would usually consist in the "**CPU HOT**" point, that is the temperature which the cpu should never exceed while controlled by the thermal governor, and the "**CPU CRIT**" point, which is the maximum allowable temperature. If the cpu exceeds for any reason the critical point the system is immediatly shut down and a message is written in the system log.

Thus, the cpu governor has to set the cpu frequency among the permitted values. The maximum and minimum frequencies change dynamically depending on the user actions and the thermal governor, the latter can only set the maximum allowable frequency based on the trend of the current cpu temperature.

In the **sysfs** there is a section for the cpufreq module, where the values **scaling_min_freq** and **scaling_max_freq** can be found. These are the interval in which the cpu governor can play a role in picking the right frequency for the system.

Testing the governor

The test consist in a batch run of many rt-app tests, each one running a different taskset. A taskset is a collection of tasks. Each task has a runtime **R** and a period **T**, with always T greater than R. We say that the governor has misbehaved if happens that some tasks cannot consume its whole runtime within its period. When a tasks consume all its runtime we call that interval T1, and when the current period ends we call that interval T2. Let us define the "**slack**" as T2-T1. It is clear that a performing system will always serve the awaiting tasks in order to leave them with a slack greater then zero. If, however, the task cannot consume its whole runtime before the period ends, the slack value would keep track of that.

By counting the number of negative slacks in a batch run we can define a metric to measure how bad a governor will behave toward performance.

Each governor will undergo several loads, in order to test many scenarios at once. Thus, the cumulative test will consist in many taskset with the utilization parameter U varying uniformly betwen 1.0 and 2.0*Cores. Of course we expect that as the taskset has a larger utilization, the number of negative slacks will increase as wells. Moreover, testing different U values allows us to actually measure the overall behaviour of the governor under many load conditions.

Results

The following section reports the result obtained from testing on a quad core ARM-7 machine, the AllWinner H2+ SoC mounted on a OrangePiZero board featuring 512 MB of DDR3 SDRAM.

The operating system is Armbian. The output of "`$uname -a`" is: "`Linux orangepizero 5.12.0-rc4+ #1 SMP PREEMPT Mon Mar 29 15:16:39 CEST 2021 armv7l GNU/Linux`"

The test can be replicated by simply:

```
$git clone https://github.com/VirtuContraFurore/Linux_governor_efficiency.git
$cd Linux_governor_efficiency
$chmod +x ./governor.sh
$sudo ./governor.sh
```

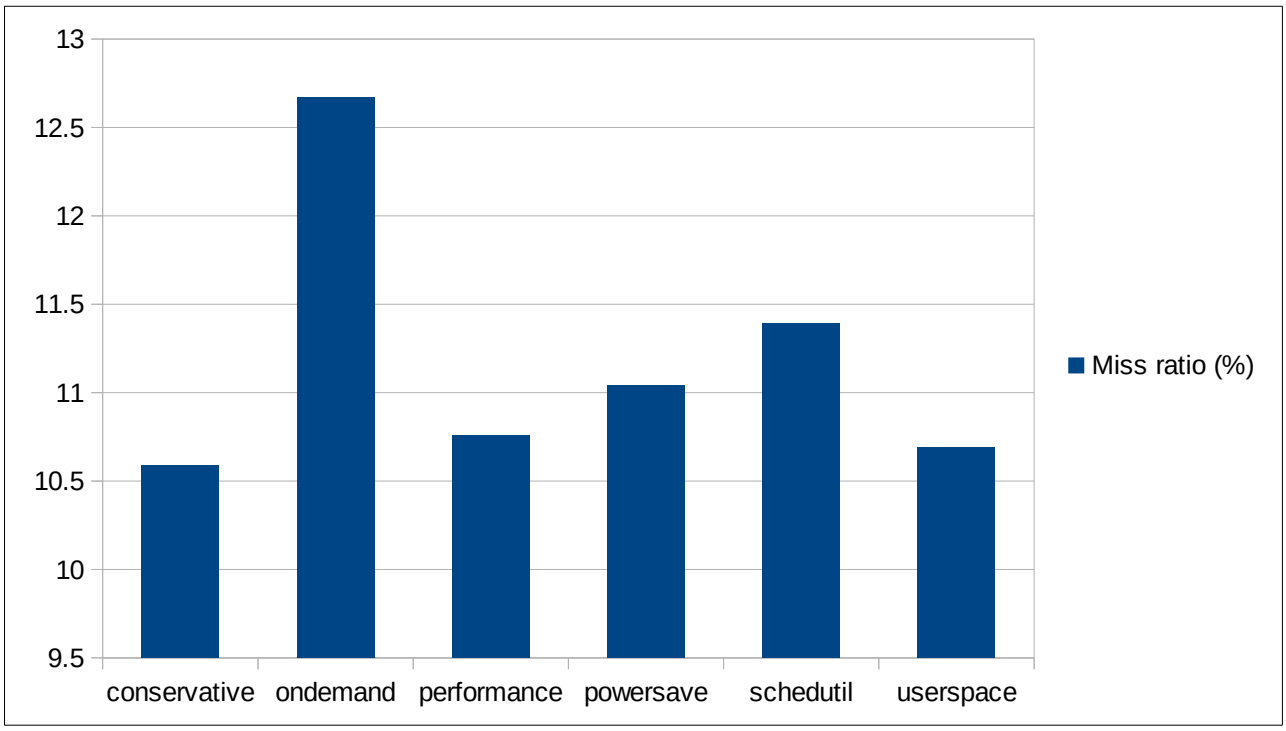
After that, the global result can be obtained by:

```
$cat ./governor-results/*.txt | grep ratio
```

Plotted data has been obtained using "**SCHED_OTHER**" as scheduling class and **2 seconds** as rt-app taskset test lenght (executing each taskset for two seconds).

Also, 10 utilization values were used, with 4 taskset each with 10 tasks.

Resulting in:



From the chart above, we can tell that **conservative**, **performance** and **userspace** governors are in the same class in terms of performance, sharing a miss ratio sightly greater than 10.6%. While **ondemand** governor performed worst, obtaing an higher miss ratio of 12.7%.

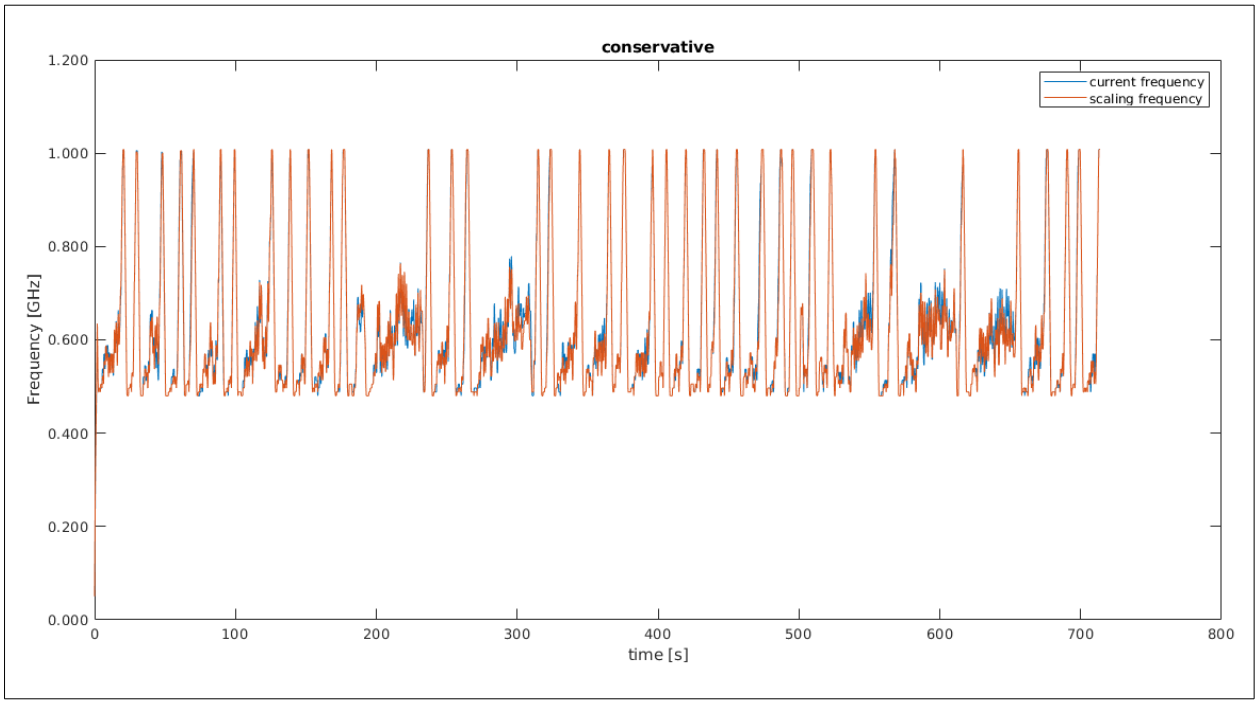
The **miss ratio** has been calculated with the formula: $miss\,ratio = \frac{negative\ slacks}{total\ slacks}$

Also, during the rt-app tests, temporal data of the CPU frequency has been logged by the "freq_logger" program, generating some reports in the form "**csv**" which stands for comma separated values.

The frequency data have been ploted using Matlab, applying a moving average filter to discard fast frequency transistions and make the plot more readable.

The **moving average filter** substitute to each value the arithmetic mean of the last N values. In this report, N = 20 has been chosen.

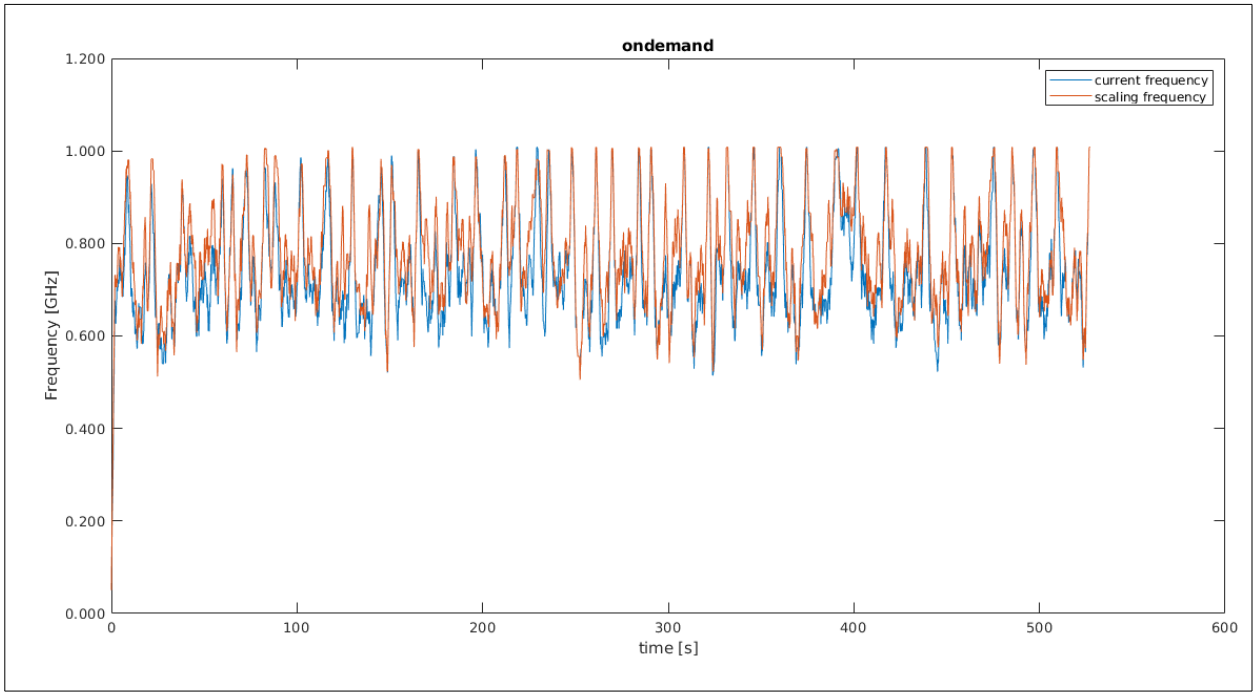
Conservative



Governor conservative: Negative slacks/total slacks: 2590/24443

Governor conservative: Miss ratio: 10.5900 %

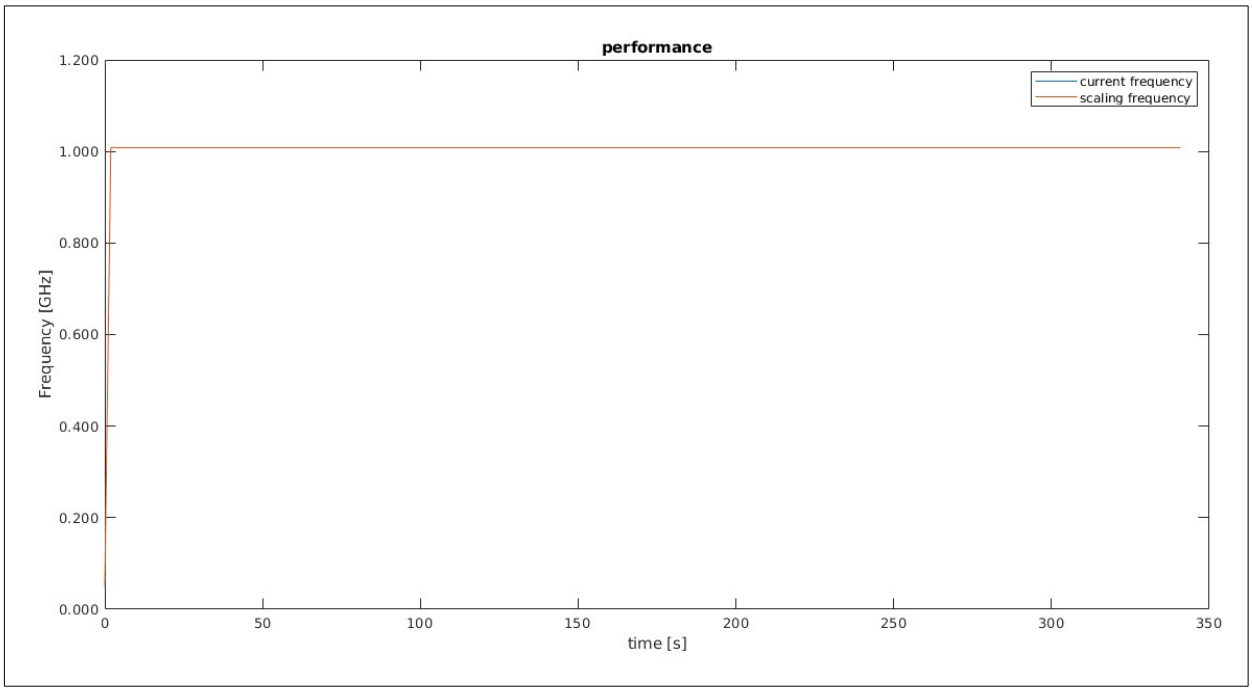
Ondemand



Governor ondemand: Negative slacks/total slacks: 3054/24086

Governor ondemand: Miss ratio: 12.67 %

Performance

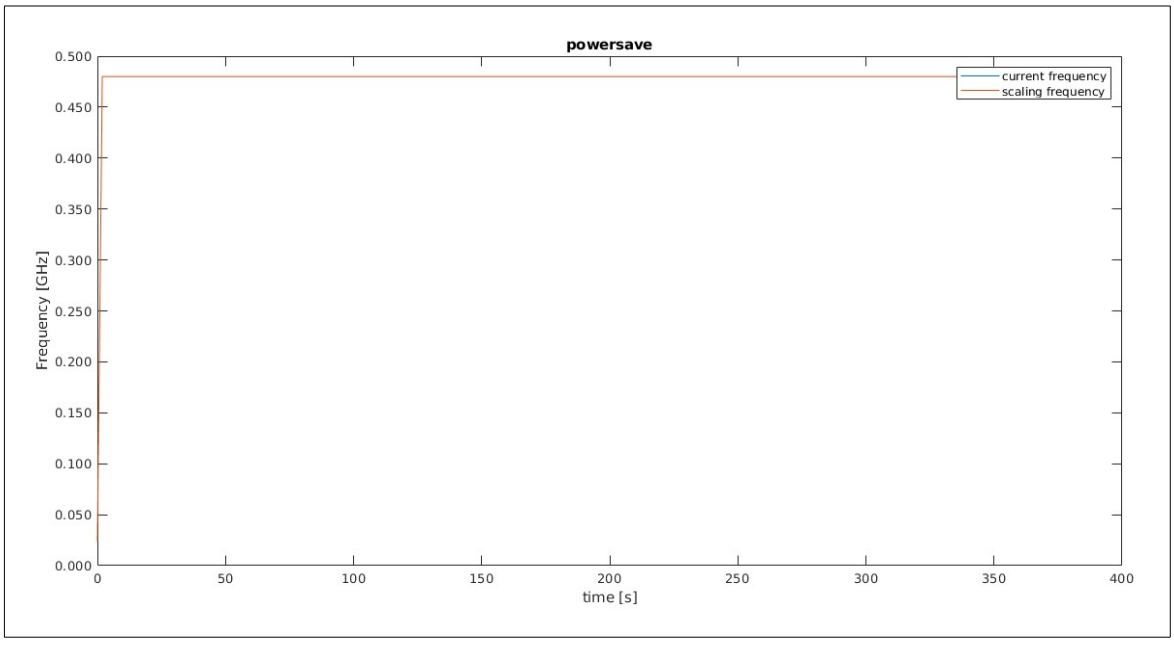


As expected, performance governor sets the frequency to the maximum allowable, 1.008 MHz.

```
Governor performance: Negative slacks/total slacks: 2618/24315
```

```
Governor performance: Miss ratio: 10.76 %
```

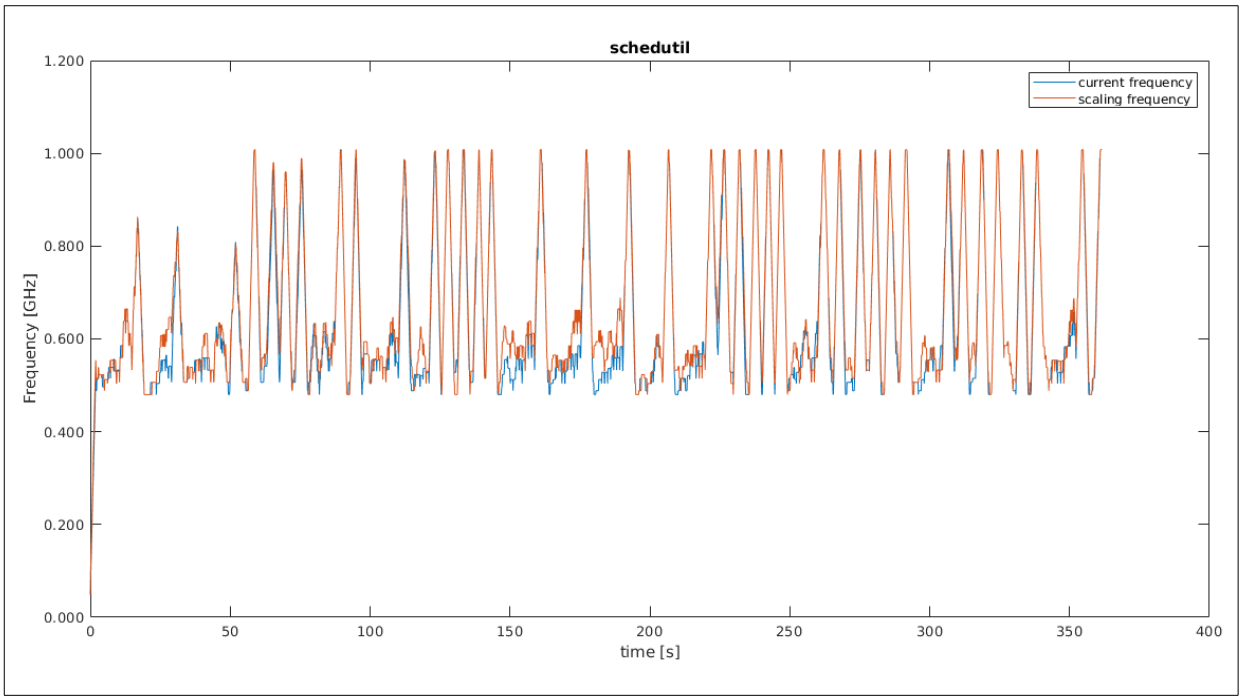
Powersave




```
Governor powersave: Negative slacks/total slacks: 2684/24298
```

```
Governor powersave: Miss ratio: 11.04 %
```

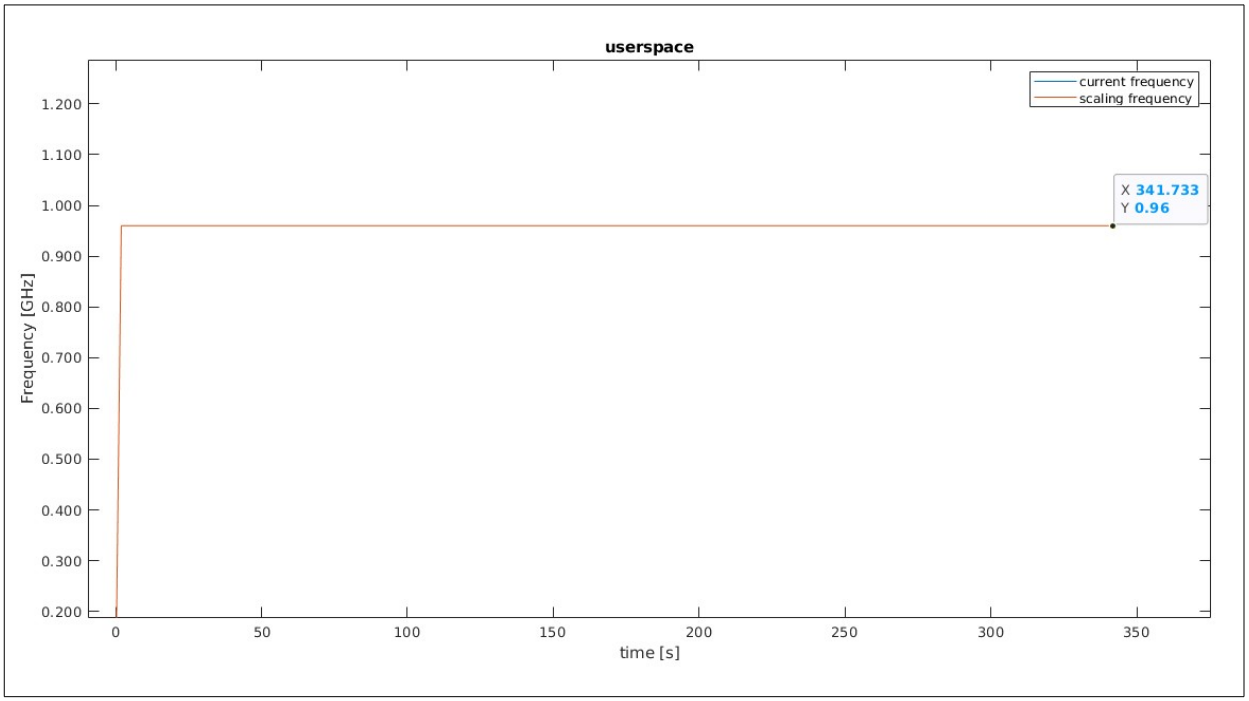
Schedutil



Governor schedutil: Negative slacks/total slacks: 2762/24243

Governor schedutil: Miss ratio: 11.39 %

Userspace

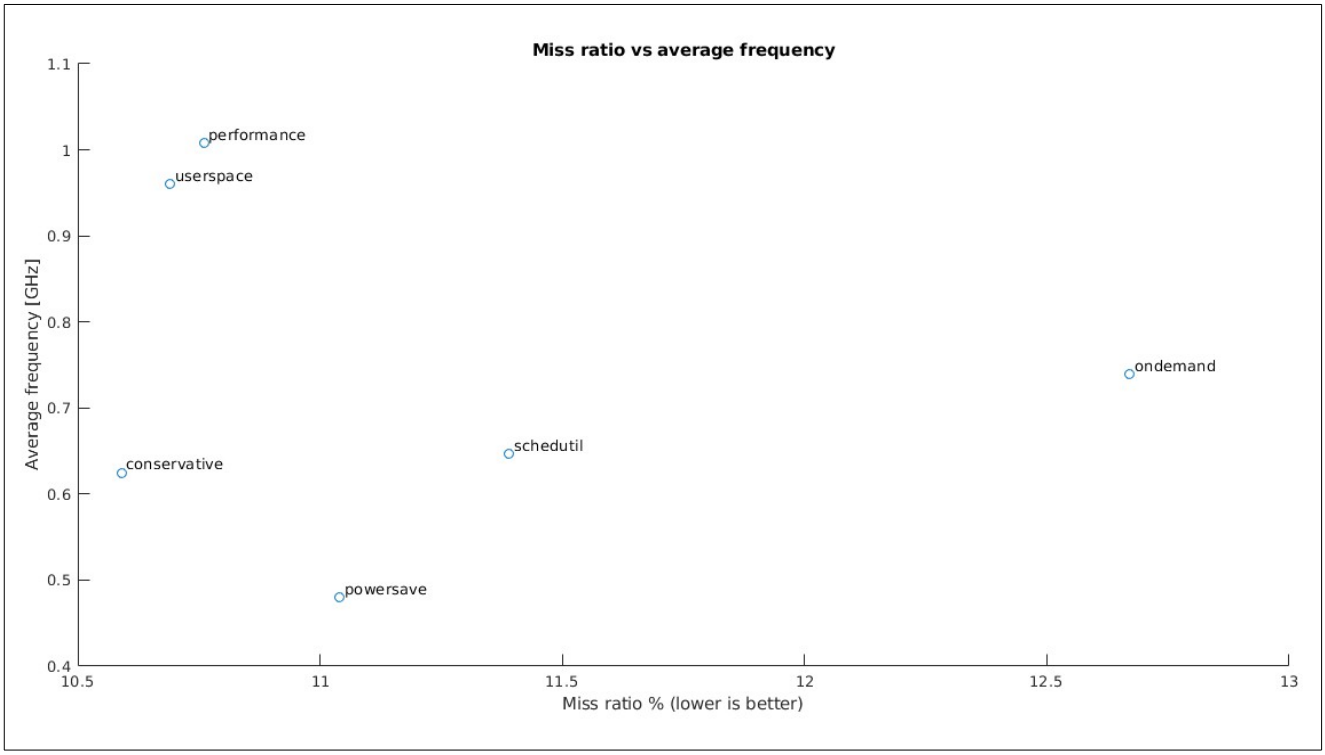


Governor userspace: Negative slacks/total slacks: 2611/24409

Governor userspace: Miss ratio: 10.69 %

Ralation between avg freq and miss ratio?

I decided to plot miss ratio vs average frequency:



It turns out that there isn't a clear relation between miss ratio and average frequency.

Future improvements

- **shuffling** the governor to test in order to make the test independent from the thermal limits of the board. Alternatively, the test suite must actively monitor the CPU temperature and wait for it to reach a low value befor beginning the test
- **shuffling** the tasksets, in order to avoid unforseen correlations between the order of the taskset and the miss ratio.
- **avoid** rt-app calibration. This can be (maybe) obtained in two ways: first one is to calibrate only once for all, and the reuse the calibration number "cycle to ns" in all the subsequent JSONS. The second way aims to avoid to use a differnt json for a different taskset, exploting rt-app flexibility, there may be a strategy to create only one big json which contains all the taskset which are now tested separately.