Adaptive Scheduling Parameters Manager for SCHED_DEADLINE

Alessio Balsini
a.balsini@sssup.it

Università di Pisa, Scuola Superiore Sant'Anna

Workshop on Real-Time Scheduling in the Linux Kernel
27 June 2014

Introduction

- Context: soft real-time periodic tasks scheduling
- Subcontext: multimedia audio/video reproduction
- Problem: tradeoff between overprovisioning and QoS
- Solution: a set of tools that manage SCHED_DEADLINE parameters adaptively

Scheduling Soft Real-Time Periodic Tasks - *What Happens*

Computational request at each activation may heavily differ.

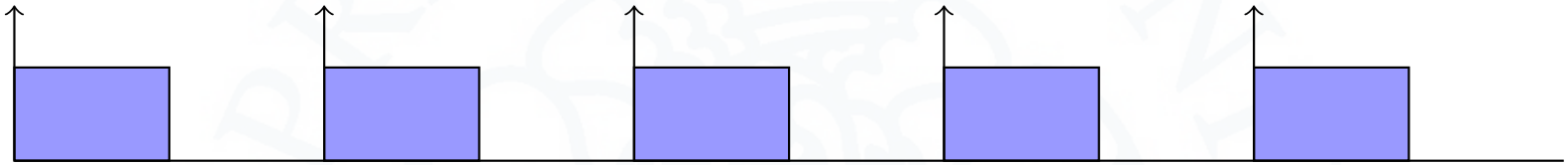


Figure : Ideal

Figure : Real

Scheduling Soft Real-Time Periodic Tasks - *Examples*

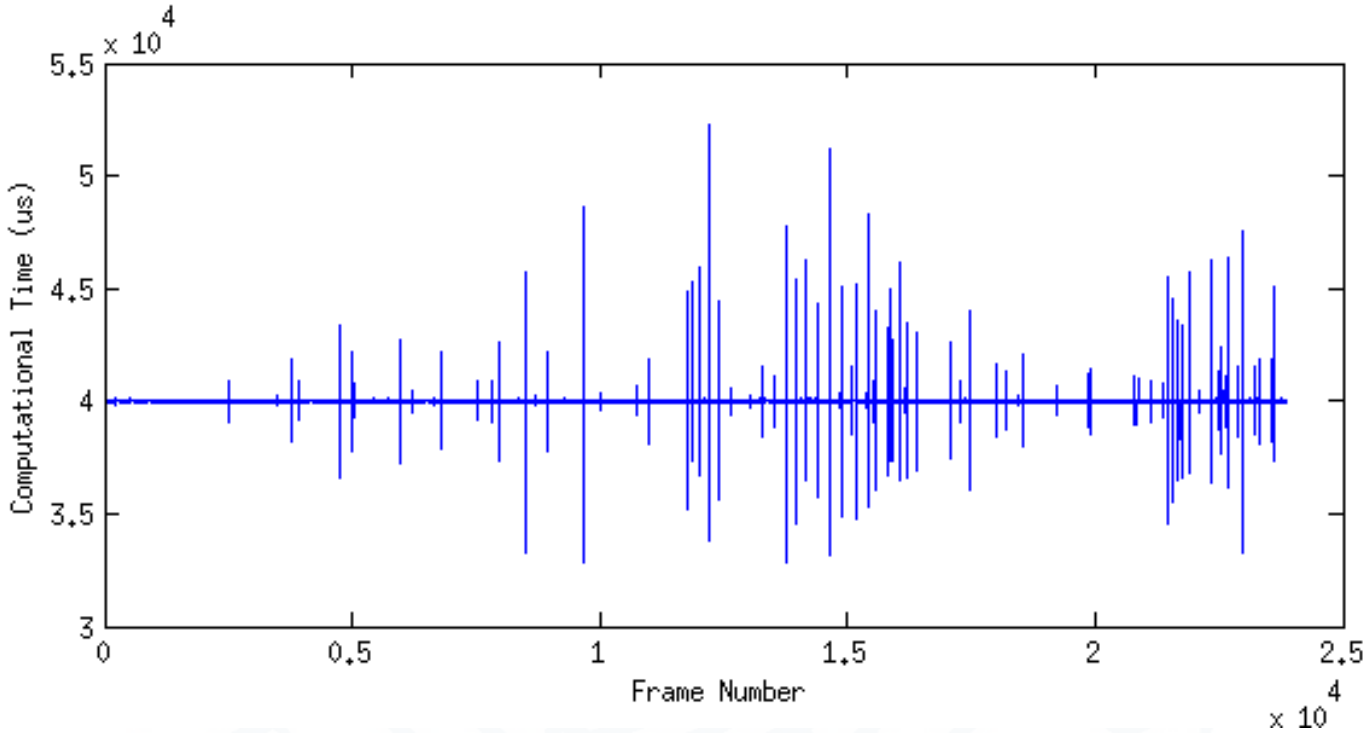


Figure : Back to the Future (MKV)

Scheduling Soft Real-Time Periodic Tasks - *Examples*

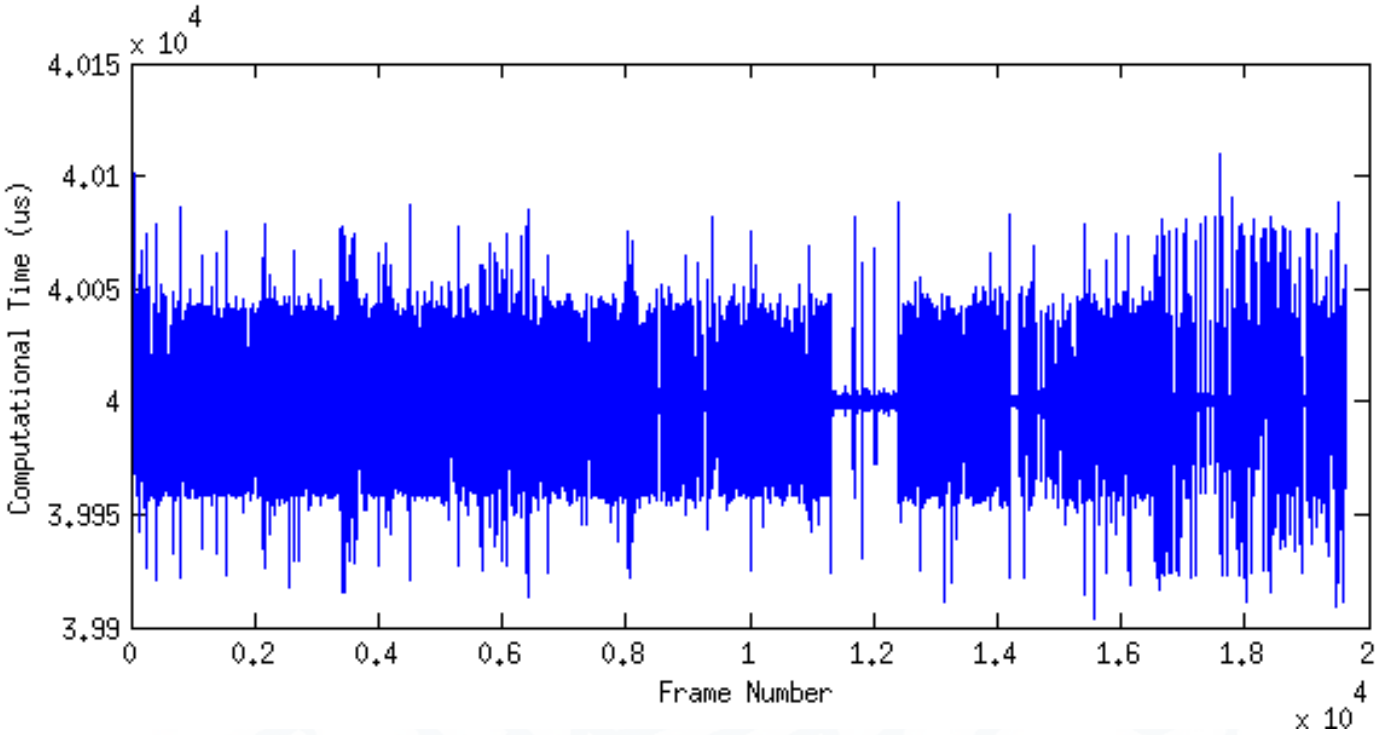


Figure : Blade Runner (AVI)

Scheduling Soft Real-Time Periodic Tasks - *Examples*

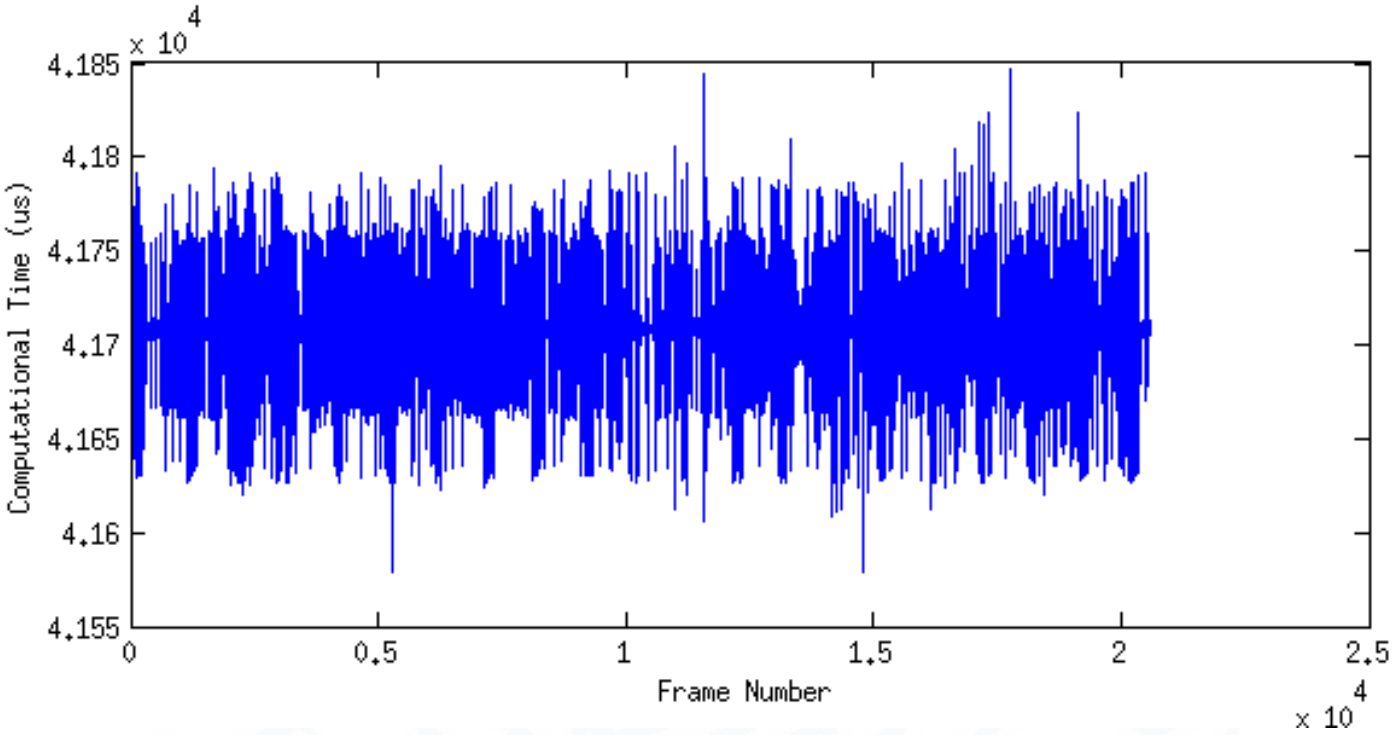


Figure : Superman Returns (MP4)

Scheduling Soft Real-Time Periodic Tasks - *Relevant Parameters?*

In SCHED_DEADLINE it is possible to configure (task based)

- Period
- Relative Deadline
- Bandwidth

However, in the considered application context, a single parameter can be enough

Response Time

Scheduling Soft Real-Time Periodic Tasks - *Relevant Parameters?*

In SCHED_DEADLINE it is possible to configure (task based)

- Period
- Relative Deadline
- Bandwidth

However, in the considered application context, a single parameter can be enough

Response Time

Problem
Response Time to SCHED_DEADLINE: $\mathbb{R} \rightarrow \mathbb{R}^3$

How to generate SCHED_DEADLINE parameters starting from the Response Time?

Problem
Response Time to SCHED_DEADLINE: $\mathbb{R} \rightarrow \mathbb{R}^3$

How to generate SCHED_DEADLINE parameters starting from the Response Time?

1. Period
   - Equal to the Response Time
2. Relative Deadline
3. Bandwidth

Problem
Response Time to SCHED_DEADLINE: $\mathbb{R} \rightarrow \mathbb{R}^3$

How to generate SCHED_DEADLINE parameters starting from the Response Time?

1. Period
   - Equal to the Response Time
2. Relative Deadline
3. Bandwidth

Problem
Response Time to SCHED_DEADLINE: $\mathbb{R} \rightarrow \mathbb{R}^3$

How to generate SCHED_DEADLINE parameters starting from the Response Time?

1. Period
   - Equal to the Response Time
2. Relative Deadline
   - Equal to the Period
3. Bandwidth

Problem
Response Time to SCHED_DEADLINE: $\mathbb{R} \rightarrow \mathbb{R}^3$

How to generate SCHED_DEADLINE parameters starting from the Response Time?

1. Period
   - Equal to the Response Time
2. Relative Deadline
   - Equal to the Period
3. Bandwidth

Problem
Response Time to SCHED_DEADLINE: $\mathbb{R} \rightarrow \mathbb{R}^3$

How to generate SCHED_DEADLINE parameters starting from the Response Time?

1. Period
   - Equal to the Response Time
2. Relative Deadline
   - Equal to the Period
3. Bandwidth
   - ?

Problem

Response Time to SCHED_DEADLINE: $\mathbb{R} \rightarrow \mathbb{R}^3$

How to generate SCHED_DEADLINE parameters starting from the Response Time?

1. Period
   - Equal to the Response Time
2. Relative Deadline
   - Equal to the Period
3. Bandwidth
   - ?

Computational Requirements
- *Is the glass half empty or half full?*

Warning: Choosing the bandwidth may cause headaches

Computational Requirements
- *Is the glass half empty or half full?*

Warning: Choosing the bandwidth may cause headaches

Optimistic

Pessimistic

Computational Requirements
- *Is the glass half empty or half full?*

Warning: Choosing the bandwidth may cause headaches

Optimistic

1. Low QoS
2. Resources-driven?

Pessimistic

Computational Requirements
- *Is the glass half empty or half full?*

Warning: Choosing the bandwidth may cause headaches

Optimistic

1. Low QoS
2. Resources-driven?

Pessimistic

Computational Requirements
- *Is the glass half empty or half full?*

Warning: Choosing the bandwidth may cause headaches

Optimistic

1. Low QoS
2. Resources-driven?

Pessimistic

1. Waste of resources

Computational Requirements
- *Is the glass half empty or half full?*

Warning: Choosing the bandwidth may cause headaches

Optimistic

1. Low QoS
2. Resources-driven?

Pessimistic

1. Best QoS
2. Waste of resources

Computational Requirements
- *Is the glass half empty or half full?*

Warning: Choosing the bandwidth may cause headaches

Optimistic

1. Low QoS
2. Resources-driven?

Pessimistic

1. Best QoS
2. Waste of resources

Another possible approach? Dynamic!
- *Let's see how much you drank*

The bandwidth is dynamically (periodically) chosen *for each SCHED_DEADLINE task*, depending on the history of the required computational times.
It is a feedback loop controller.

But if this controller modifies the bandwidth, isn't it just like removing the CBS from SCHED_DEADLINE?

Yes and no.

- Adaptation of Q_i and P_i
- Global controller

Another possible approach? Dynamic!
- *Let's see how much you drank*

The bandwidth is dynamically (periodically) chosen *for each SCHED_DEADLINE task*, depending on the history of the required computational times.
It is a feedback loop controller.

But if this controller modifies the bandwidth, isn't it just like removing the CBS from SCHED_DEADLINE?

Yes and no.

- Adaptation delays for the transitory
- Global controller

Another possible approach? Dynamic!
- *Let's see how much you drank*

The bandwidth is dynamically (periodically) chosen *for each SCHED_DEADLINE task*, depending on the history of the required computational times.
It is a feedback loop controller.

But if this controller modifies the bandwidth, isn't it just like removing the CBS from SCHED_DEADLINE?

Yes and no.

- Adaptation delays for the transitory
- Global controller for the fairness

Another possible approach? Dynamic!
- *Let's see how much you drank*

The bandwidth is dynamically (periodically) chosen *for each SCHED_DEADLINE task*, depending on the history of the required computational times.
It is a feedback loop controller.

But if this controller modifies the bandwidth, isn't it just like removing the CBS from SCHED_DEADLINE?

Yes and no.

1. Adaptation delay: for the transitory
2. Global controller: for the fairness

Another possible approach? Dynamic!
- *Let's see how much you drank*

The bandwidth is dynamically (periodically) chosen *for each SCHED_DEADLINE task*, depending on the history of the required computational times.
It is a feedback loop controller.

But if this controller modifies the bandwidth, isn't it just like removing the CBS from SCHED_DEADLINE?

Yes and no.

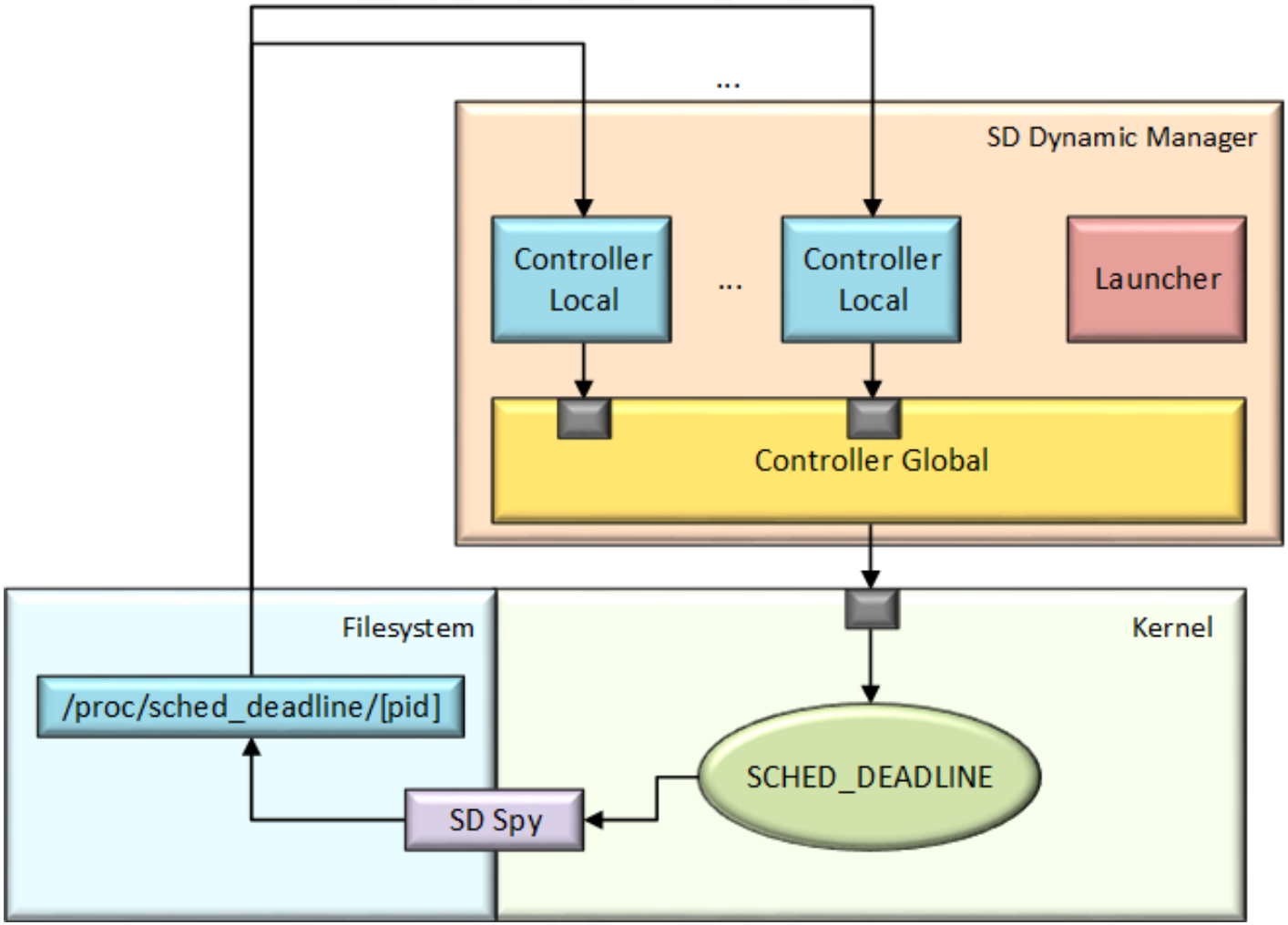
1. Adaptation delay: for the transitory
2. Global controller: for the fairness

Tools

Tools

- **Kernel Module**: SCHED_DEADLINE Spy
- **Daemon**: SCHED_DEADLINE Dynamic Manager
- **Configuration GUI**: SchedConfigTool

Hi-level Point of View
- *Overall Block Scheme*



Kernel Module: SCHED_DEADLINE Spy - *Userspace*

This module creates a file for each SCHED_DEADLINE task

/proc/sched_deadline/[PID]

Containing four columns:

```
1401468028 22757242 10149132 N
1401468028 22757242 86353 N
1401468028 62757243 37679835 Y
1401468028 94757243 26311134 N
...
```

- first two columns are the kernel time (seconds and nanoseconds) of the measurement
- third column is the job execution time (nanoseconds)
- last column says if execution exceeds the bandwidth (*Yes*/*No*)

Kernel Module: SCHED_DEADLINE Spy - *Implementation Hints*

Kernel probes: *Kprobe*s
Probes are placed around the kernel and the instrumentation codes are executed when the processor encounters those probe point. With *Jprobe*s it is also possible to access function arguments.
This module attaches probes to

enqueue_task_dl

and

update_dl_entity

*Jprobe*s are used to create tasks' statistics.
Other callbacks are provided, managing all the statistics sequential files.

Daemon: SCHED_DEADLINE Dynamic Manager - *Interface*

This tool provides the following DBus interface

core.sched.dl.ProcessManager

with the following methods:

- **xml**: requires a string input, corresponding to the path of the XML file containing the task information
- **fixed_add**: adds a new fixed task to the control list, with the defined SCHED_DEADLINE parameters
- **fixed_launch**: creates a new fixed task and adds it to the control list, with the defined SCHED_DEADLINE parameters
- **control**: adds a new dynamic task to the control list, with the defined response time parameter
- **launch**: creates a new dynamic task and adds it to the control list, with the defined response time parameter

Daemon: SCHED_DEADLINE Dynamic Manager - *Interface, Sample Configuration*


```
<?xml version="1.0"?>
<SchedulingAlgorithm name="SCHED_DEADLINE">
  <path>/usr/bin/executable</path>
  <args>-p parameter</args>
  <runtime>28000000</runtime>
  <deadline>33333333</deadline>
  <period>33333333</period>
</SchedulingAlgorithm>
```

XML configuration for SCHED_DEADLINE Dynamic Manager

Daemon: SCHED_DEADLINE Dynamic Manager - *Implementation, Controller Local*

- One controller local for each dynamically scheduled task
- It performs the following operations cyclically
  - obtain task statistics
  - run the Control Algorithm to calculate the best utilization factor
  - send the computed utilization factor to the global controller

Note: The current control algorithm implements the worst case within a window of samples

Daemon: SCHED_DEADLINE Dynamic Manager - *Implementation, Controller Global*

It performs the following operations cyclically

- check the schedulability of all the SCHED_DEADLINE utilization factors

$$\sum_{i=1}^{n} \frac{B_{D,i}}{T_{D,i}} + \sum_{i=1}^{m} \frac{B_{F,i}}{T_{F,i}} \leq B_{SD}$$

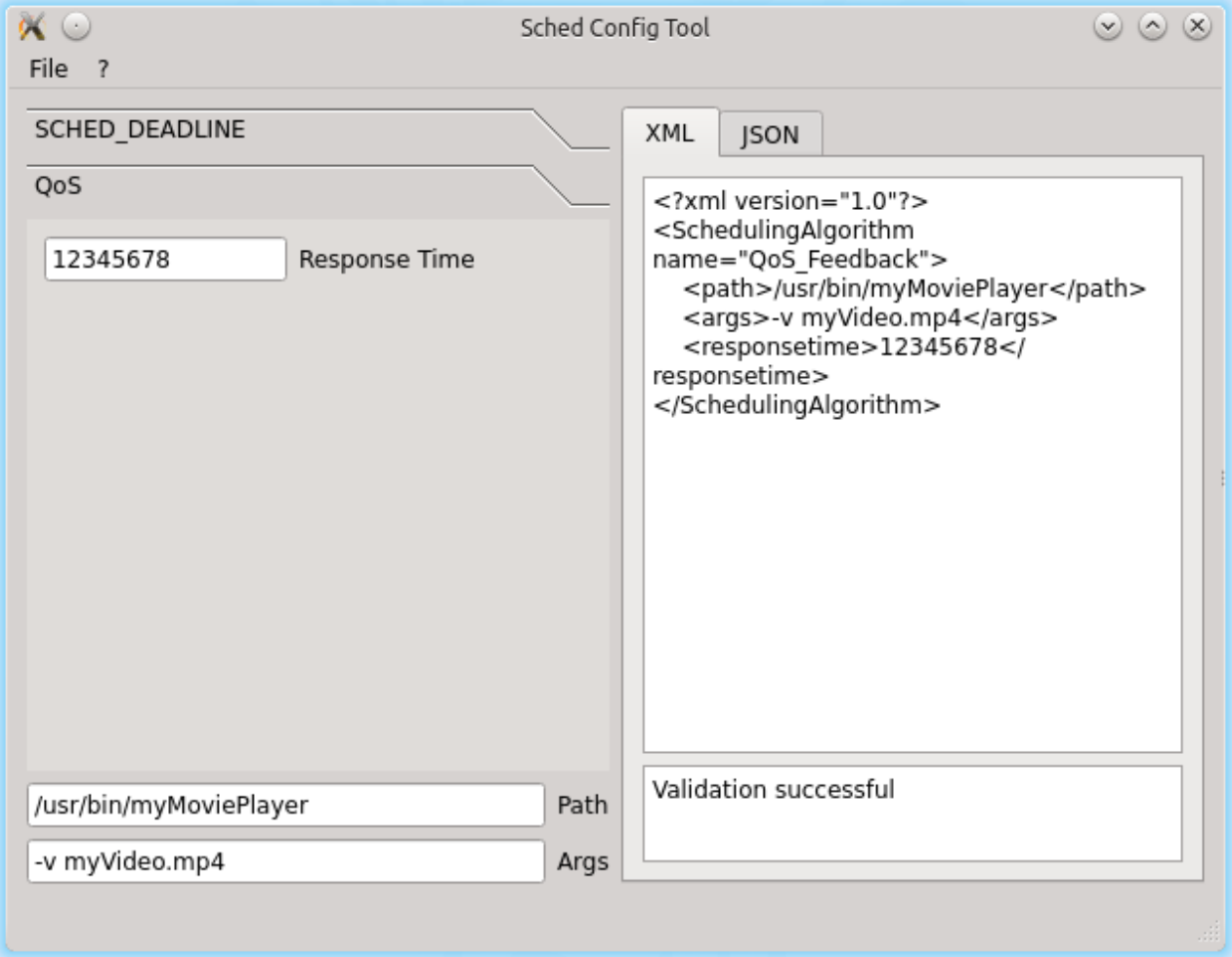
  D: Dynamic, F: Fixed

  - if not verified, use the *Spring With no Length Constraints* algorithm to compress the dynamic tasks' requirements

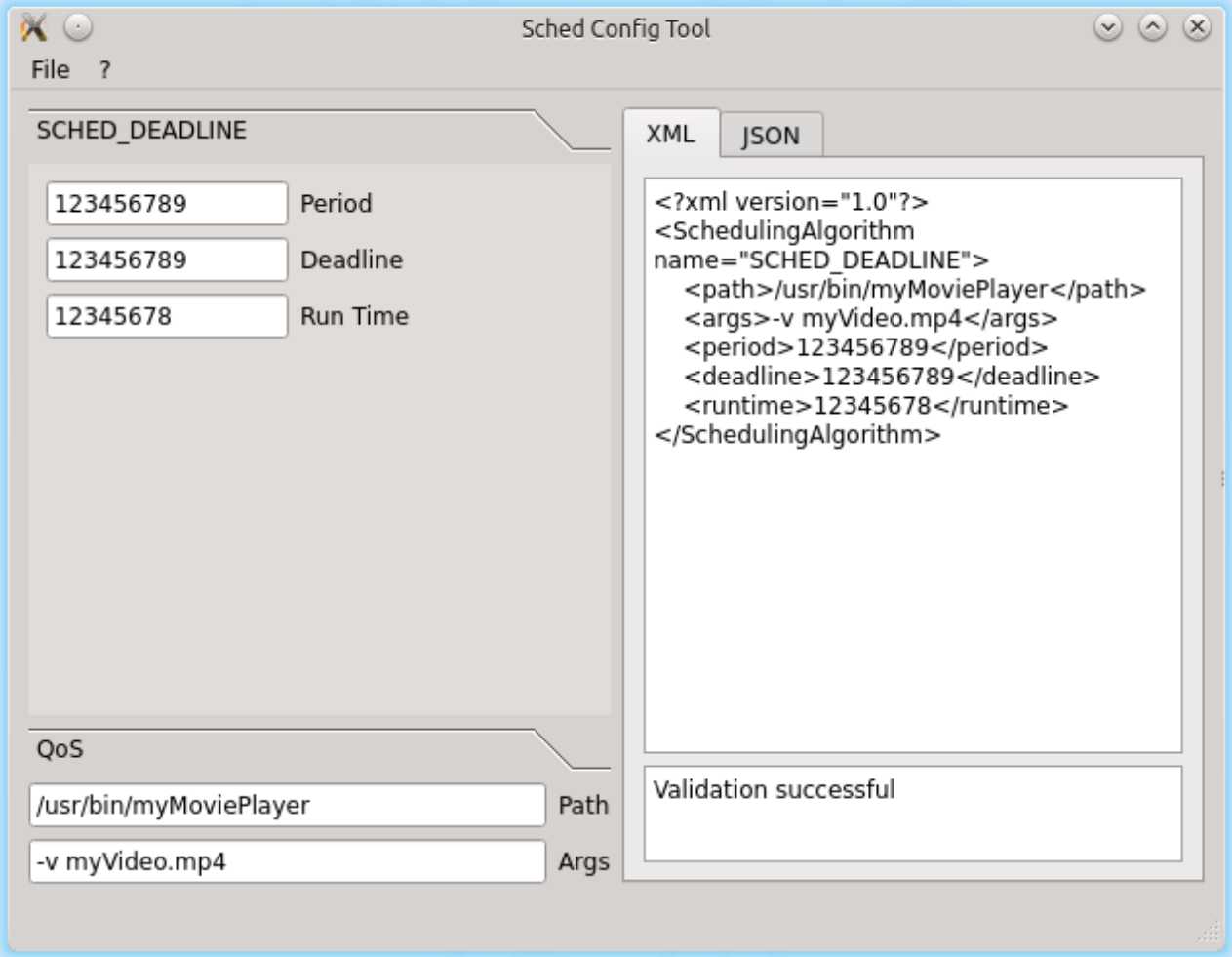
$$\forall i,\ U'_{D,i} = \frac{B_{D,i}}{T_{D,i}} - (U_D - B_{residual}) \cdot \frac{T_{D,i}}{\sum_{i=1}^{n} T_{D,i}}$$

- update SCHED_DEADLINE parameters

Configuration Generator: SchedConfigTool



Configuration Generator: SchedConfigTool



Performance

Response Times
- *MPlayer Without SCHED_DEADLINE*

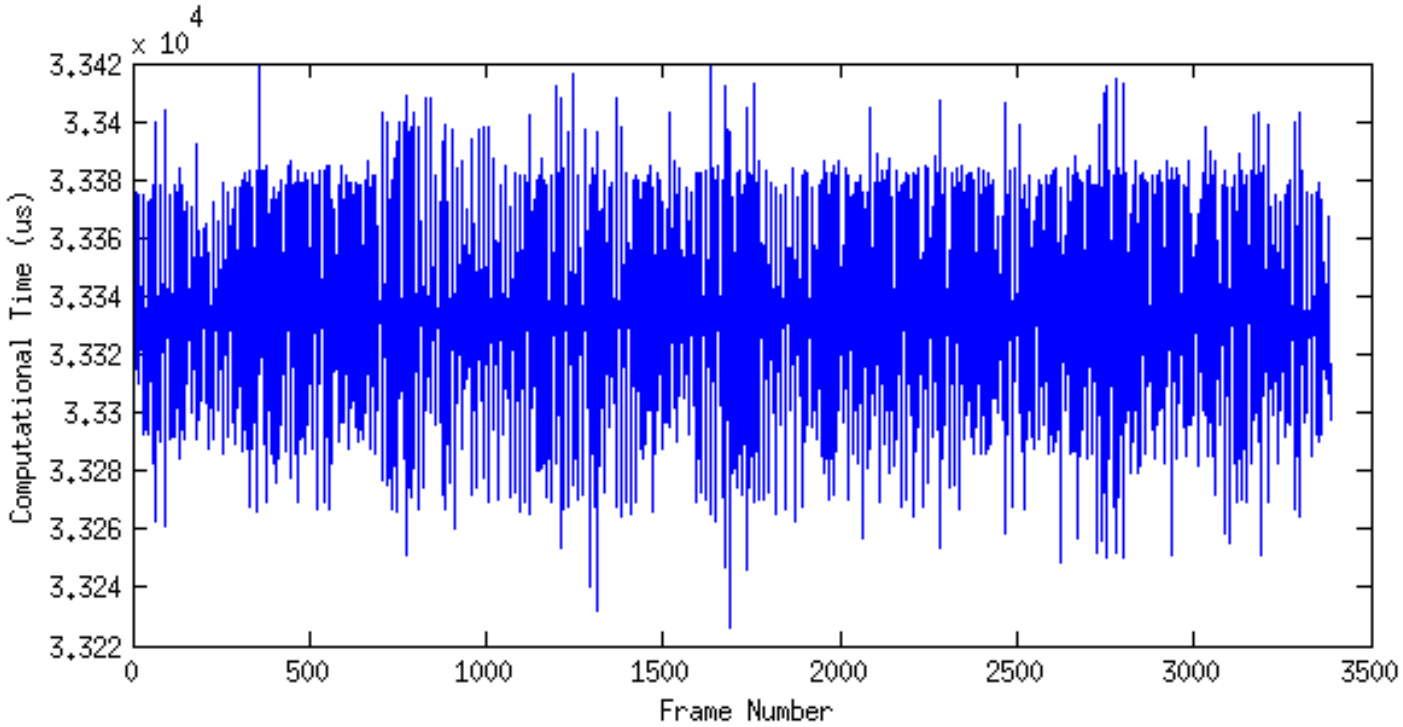


Figure : Eve Online Rubicon (MP4) without SCHED_DEADLINE

Response Times
- *MPlayer With Dynamic Manager, Alone*

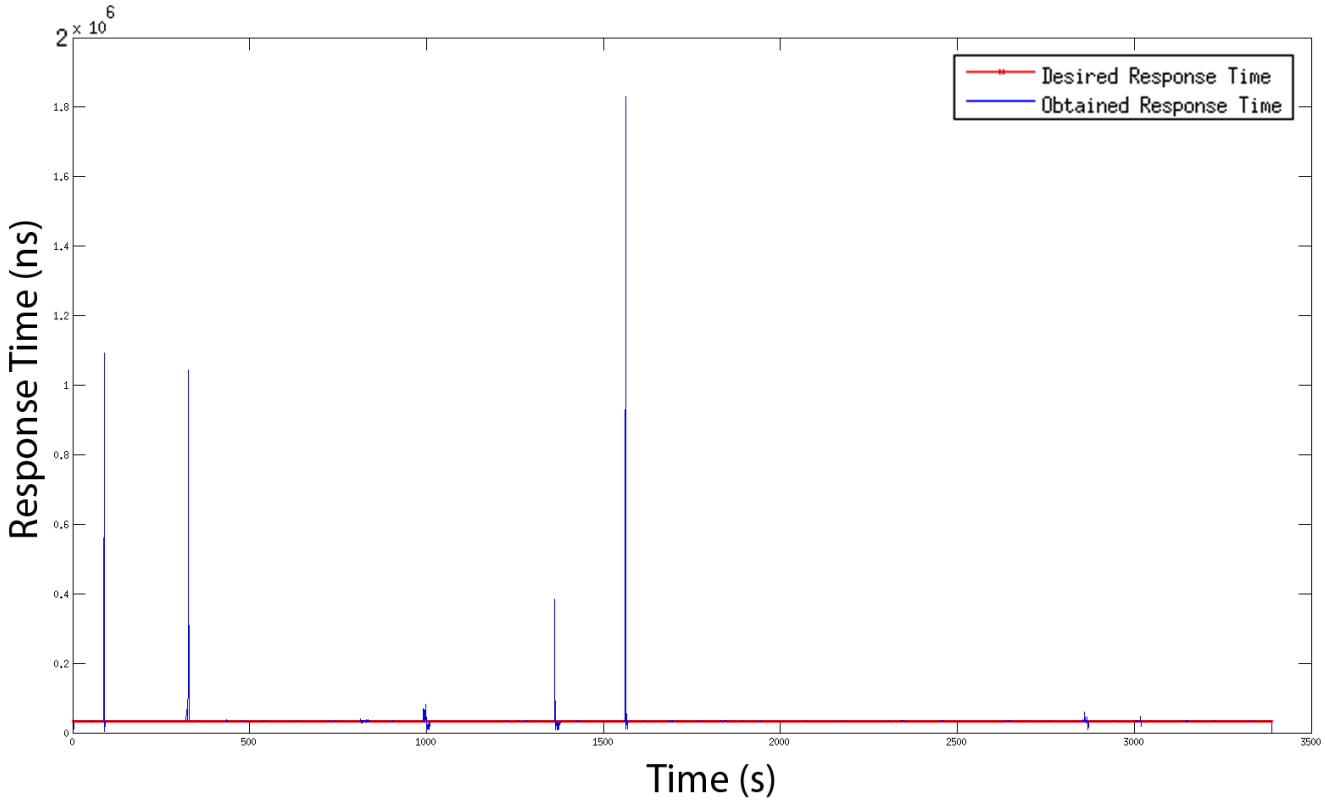


Figure : Eve Online Rubicon (MP4) Dynamic Manager, Alone

Controller Global period: 1s, Controller Local window size: 50

Response Times
- *MPlayer With Dynamic Manager, With Fixed*

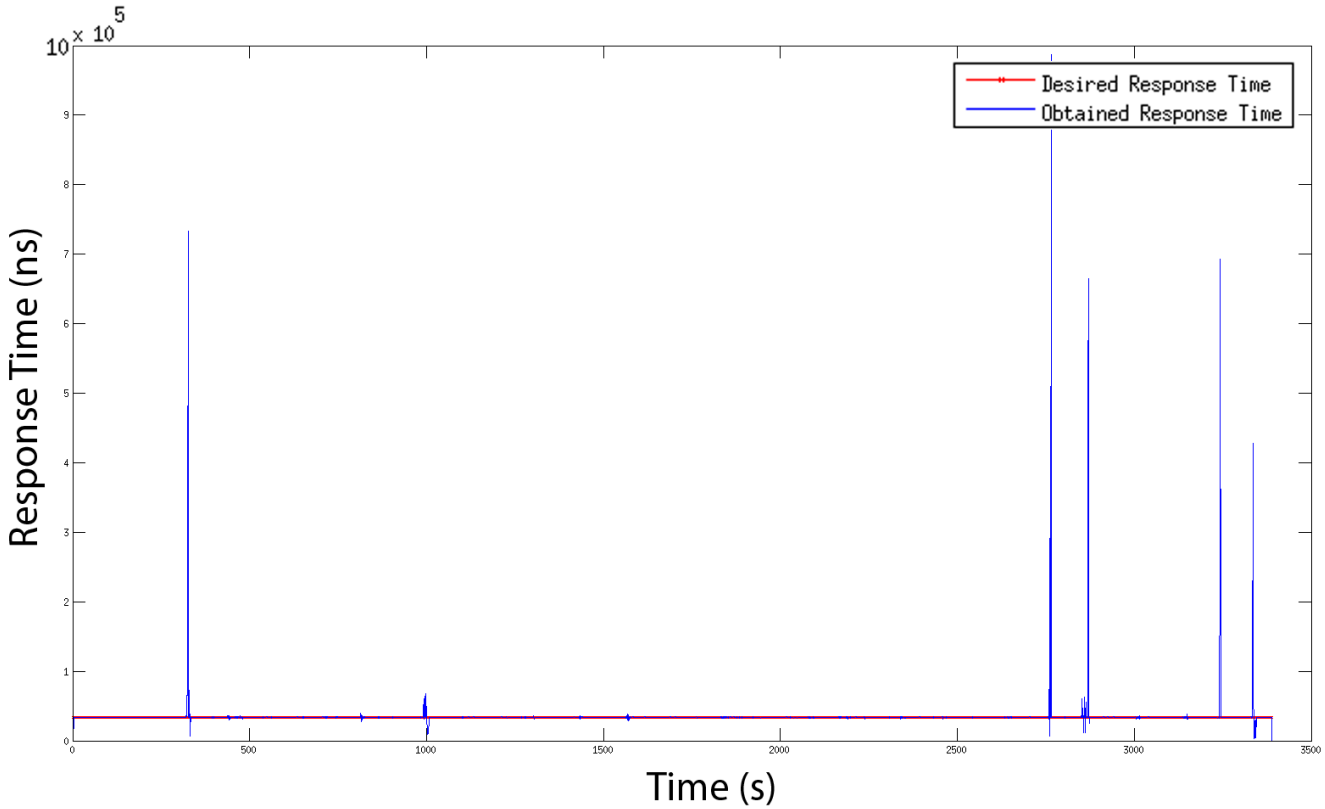


Figure : Eve Online Rubicon (MP4) Dynamic Manager, Running Together With Several Fixed Parameters SCHED_DEADLINE Tasks

Controller Global period: 1s, Controller Local window size: 50

Response Times
- *MPlayer With Dynamic Manager, With Other*

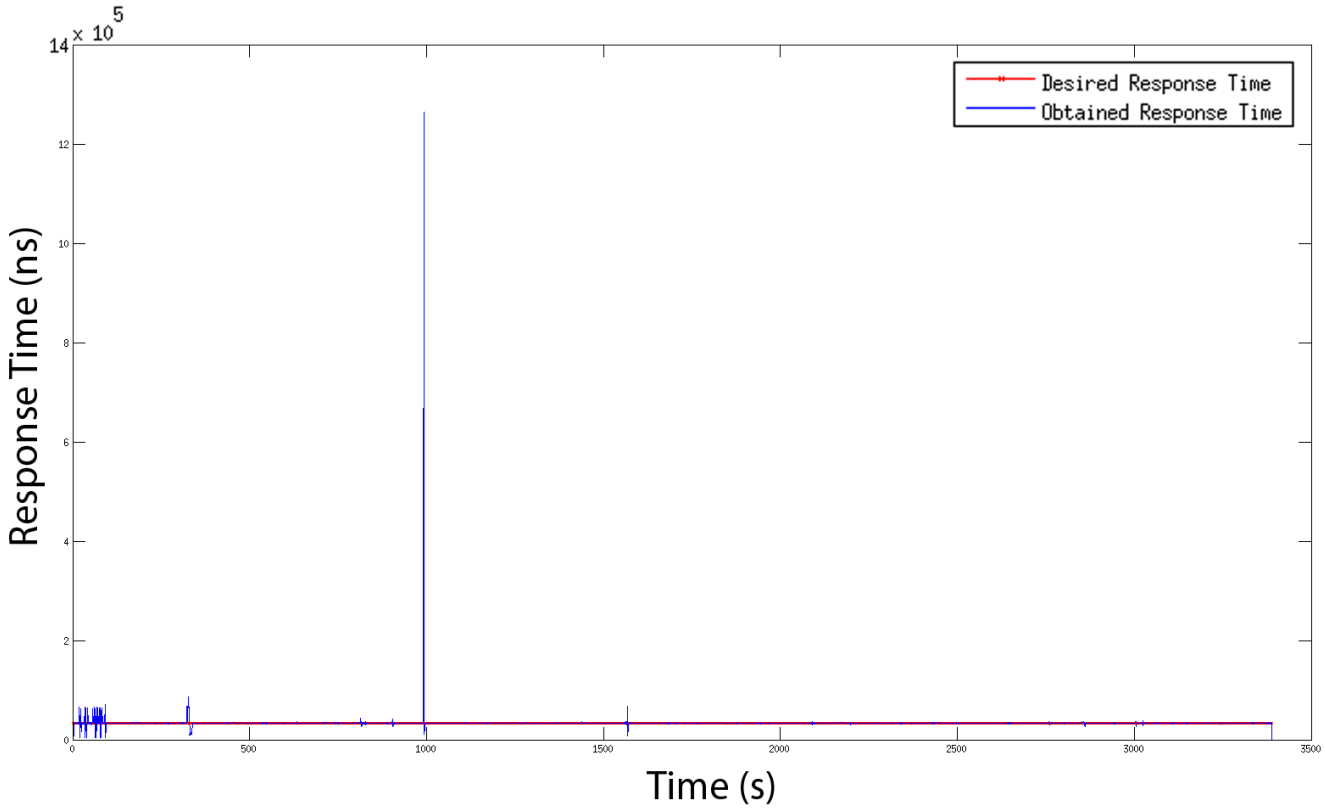


Figure : Eve Online Rubicon (MP4) Dynamic Manager, Running Together With Several Other Linux Tasks

Controller Global period: 1s, Controller Local window size: 50

Practical Session

Repositories

- SCHED_DEADLINE Spy
  - github.com/balsini/sched-deadline-spy
- SCHED_DEADLINE Dynamic Manager
  - github.com/balsini/sched-deadline-dynamic-manager
- SchedConfigTool
  - github.com/balsini/SchedConfigTool

References

L. Palopoli, T. Cucinotta, L. Marzario, G. Lipari, *AQoSA - Adaptive Quality of Service Architecture*. Wiley InterScience, 2008.

G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications 3rd*. Springer Publishing Company, 2011.

Thank You

Alessio Balsini

✉

a.balsini@sssup.it