Implementation of new feasibility interval analysis methods in Cheddar

Université de Bretagne Occidentale
Yann Allain

***Abstract*—Since several years researchers in real-time systems are trying to figure out simulations intervals for various schedulers with multiple constraints. This work presents new calculations algorithms implementation for the feasibility intervals in Cheddar scheduling analysis tool to improve or complement the current algorithms. The solution was implemented and tested on Cheddar which is programmed in Ada language and gives corrects intervals. After several experimentations on multiples tasks sets, the solutions brings consequently a schedulability proof by simulation on multiple cases and even improves the maximum interval in contrary of the pre-existing interval calculation algorithms into Cheddar.**

***Keywords*—Real-Time Scheduling, Simulation, Cheddar, ADL.**

I. Introduction

Schedulability analysis methods allow designers to investigate how the tasks will be scheduled on a specific hardware platform and how tasks will be delayed due to hardware contentions. These methods derive from various calculations on systems characteristics. Nowadays, more and more scheduling analysis tools are appearing in the field, each of them targeting specific audiences and having specifics functionalities.

In this article, we address the issue of providing a schedulability proof for a given task set and a given hardware platform using scheduling simulation in Cheddar scheduling analysis tool. To fulfill the objective of giving a proof of scheduling via simulation, the path envisaged to achieve this requires the implementation into the Cheddar framework [1] of simulation intervals bounds based on the materials provided by the paper of J. Goossens, E. Grolleau and L. Cucu-Grosjean from 2016 "Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms" [2]. Indeed, the schedulability of a system can be validated or not by running a simulation on the relevant interval [3].

A. Problem Statement

Many scheduling analysis tools exist both leading to running algebraic or analytical functionalities like feasibility tests. Some of them using model-checking in a way to analyze the reachable system states and bring a validation on the schedulability of the system. Moreover, some tools implement scheduling simulations leading to the computation of the scheduling timelines and allowing their analysis. Cheddar is one of these tools and is implementing both scheduling simulation and algebraic/analytical functionalities. It did not take into account the simulation as a proof on the schedule on the contrary of the feasibility tests. Indeed, by using Cheddar simulation there is no promise that the considered time interval will provide the evidence that the system is schedulable even if the time interval is large. Furthermore, by using huge tasks sets with big integers as tasks periods it is even more difficult to determine a time interval for the simulation that is suitable in order to have an idea of the system behavior. Considering those difficulties, we investigate how to formalize and implements simulations intervals in the Cheddar framework that will provide for various systems, a behavioral insurance of the considered system within the time interval and so, a proof on its schedulability.

B. Contributions

We first propose a new implementation of simulation intervals algorithms for Cheddar framework based on multiple papers who are bringing proof of schedulability within those time-intervals for differents systems with various characteristics.

The second contribution is the integration inside the framework of the solution which includes a way to express big integers and carry out calculations on those numbers based on a package [4]. Those contributions have been validated by multiple tasks set for each simulation interval in order to verify the behavior of the algorithms and their correctness. The analysis tool, that implements this solution allows us to run a simulation for a specific scheduler with a task set and if taken into account by the solution, gives the simulation and conclude on the schedulability of the considered system.

The remainder of the paper is as follows. The next section presents related works. Section III introduces background elements about the simulation intervals that we consider. Then, section IV proposes our simulation model of interval analysis. Implementation and evaluation of the solution are explained in section V and section VI concludes the article.

II. Related works

An interesting point to state is the work carried out by different researchers and/or developers on the implementation of this type of proof, on the same kind of validation and scheduling simulation software. After researching other tools, some of them shown similar functional characteristics in terms of simulation or validation like MoSaRT [5] or others even but none information have been found concerning the integration of multiple simulation intervals depending on the considered system to conclude on similar works. However, excluding software based projects, [2] provides a survey on feasibility intervals that we can use to implement the functionality into Cheddar.

III. Cheddar Software Environment

In this section, we introduce the software environment Cheddar within wich the solution will be implemented as well as the definitions necessary to this work.

A. Cheddar Scheduling Analysis Tool

The solution is implemented into Cheddar which is a GNU GPL framework that provides various functionalities for scheduling analysis as schedulability tests, a scheduling simulator as well as various features related to the design and the scheduling analysis of real-time systems.
In order to perform scheduling analysis, Cheddar handles an architecture design language (ADL) called *CheddarADL* [6]. This ADL allows the user to describe both hardware and software parts of the systems targeted for the analysis. Cheddar also implements its own domain specific language (DSL) inside its scheduling simulator that allows users to design new task models or scheduling policies. We will see later in section VI that we will use this DSL in order to represent our systems with different tasks set to test the various intervals implemented in our solution.

B. Intervals

1) Feasibility interval: is the first interval to be considered:

Definition 1: The feasibility interval is a finite interval $[a, b]$ such that if all the deadlines of jobs released in the interval are met, then the system is schedulable. [2]

2) Simulation interval: is the second interval that has to be considered for this work:

Definition 2: The simulation interval is a safe time interval such that the schedule repeats in a cycle. It is useful to capture the whole behavior of the system to characterize various metrics and evaluate during the simulation if the system can be considered as schedulable. Moreover knowing the length of the simulation interval is required when building a pre-run-time schedule known as offline schedule [7].

C. Notations

In order to present later the intervals calculations intervals used as others definitions and vocabulary used to present the various characteristics of the systems that will be used in the implementation of the solution some additional definitions are introduced based on [2]:

A system $S = \{\tau_1, \ldots, \tau_n\}$ including n tasks defined as follows:

$\tau_i = \{ O_i, C_i, T_i, D_i, P_i \}$

where

- $O_i \in \mathbb{N}$ the task offset, i.e., the release date of the first job $\tau_{i,1}$ of τ_i.
- $C_i \in \mathbb{N}$ specifies the Worst-Case Execution Time (WCET), i.e., the maximum amount of time required on a processor ρ for a job τ_i to be computed.
- $T_i \in \mathbb{N}$ the period of the task, i.e., the fixed delay between two jobs of the task where the jobs are released at the instants $O_i + kT_i$, $k \in \mathbb{N}$ and k the k^{nth} job of the task τ_i.
- $D_i \in \mathbb{N}$ the deadline such that the deadline of a task should be less or equal to it's period. We can note that if the system have deadlines are less than periods then the system deadlines are said *constrained* else if they are equals to period the system deadlines are said *implicit* else they're said to be *arbitrary*.
- P_i the fixed priority of the task (if in fixed task priority scheduling).

  Some other terms are also needed to understand how the intervals will be implemented and which one will be used based on the system properties.

  Tasks are said *independent* if the executions of jobs of different tasks are not related to each other in contrary of both *prececence constraints* where the execution of a job τ_j cannot be started before the end of a task τ_i so that τ_i precede τ_j as well as the *mutual exclusion constraint* when two tasks τ_i and τ_j share a critical resource and where their critical sections exclude each other.

IV. The simulation interval model

In this section, we introduce our model for the simulation intervals algorithms. We also present the way we will implement those algorithms and where and how they will be integrated into the Cheddar framework. A simple example of a tasks system will be presented and how this interval is calculated. Finally, we explain how to compute a specified interval based on the system characteristics.

A. Presentation of the approach

Figure 1 shows the overall approach we propose in this article and particularly where the solution will be integrated into the framework. Based on a system specified as an entry of the framework, that can be represented in various formats like *CheddarADL or DSL proper to Cheddar or AADL*. Then, when the simulation will be launched, the module that we implemented will be called to calculate the simulation interval and will allow the user to be aware of the validation or not of his simulation as a proof. In the sequel, we introduce a task system and the issues that we face in his representation and the calculations needed to render the simulation interval associated.

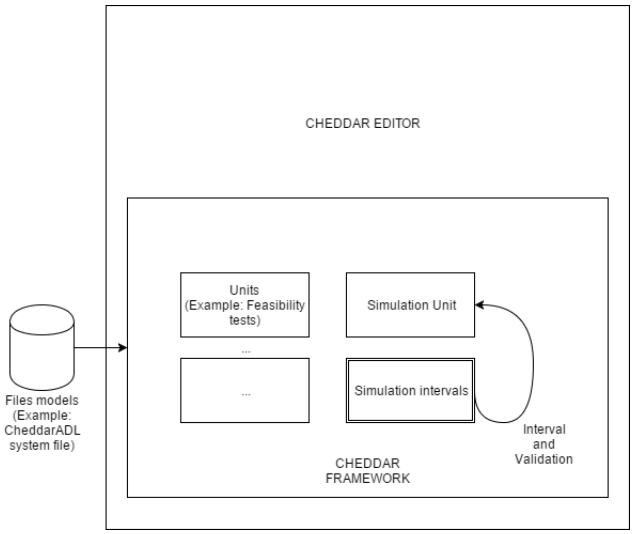

Fig. 1. Simple visualisation of the solution implementation in Cheddar

B. A simple example

We define the following example that illustrates a task model and shows how we can generate the corresponding simulation interval as well as the issues that can appear in the implementation and that have to be taken into account. For this example, we use a simple system for the representation of three periodic tasks with arbitrary deadlines that are independent on identical processors with a global fixed-task priority scheduling algorithm.

τ_i	O_i	C_i	T_i	D_i	P_i
τ_1	1	1	4	4	2
τ_2	0	2	4	5	1
τ_3	0	4	4	4	3

TABLE I. Example task set

Table 1 presents the task model. As shown in Table 1, the task τ_2 have her deadline greater than her period. All the others characteristics of the task set is represented in the table.

From this task model we can compute the simulation interval for arbitrary deadlines systems on identical multiprocessor platforms. The interval $\hat{S}_n + H$ is calculated following this way:

$$\hat{S}_1 = O_1$$
$$\hat{S}_i = max\left(O_i, O_i + \left\lceil \frac{\hat{S}_{i-1} - O_i}{T_i} \right\rceil T_i\right) + Hi$$

Where H_i the lcm of the tasks from 1 to the current task i.e., $H_i = lcm_{j=1..i}(T_i)$

So we have the following calculation:

$$\hat{S}_3 = 0$$
$$\hat{S}_1 = max\left(1, 1 + \left\lceil \frac{-1}{4} \right\rceil 4\right) + 4 = 5$$
$$\hat{S}_2 = max\left(0, 0 + \left\lceil \frac{5}{4} \right\rceil 4\right) + 4 = 12$$
$$\hat{S}_2 + H = 16$$

C. Decision Tree

Now that we've seen one of the intervals and it's calculation we can extrapolate properties and verification that we will have to implement in the program. Indeed, we know that there is a need to dispatch the function calls to a specific simulation interval algorithm [2] based on the system properties. Consequently as shown in Figure 2 we can define a decision tree showing the multiple possibilities. Here are the abbreviations and their meaning:

- MN : monoprocessor
- UN : uniform
- UR : unrelated
- ID : identical
- AD : arbitrary deadlines

- CD : constrained deadlines
- IND : independent tasks
- MUX : mutual exclusion dependency
- PRE : precedency dependency
- FJ : fixed-job priority scheduler
- FT : fixed-task priority scheduler
- GFT : global fixed-task priority scheduler

- 1 : $[0, O^{max} + 2H)$
- 2 : $[0, S_n + H)$
- 3 : $[0, \theta_c + H)$
- 4 : $[0, \hat{S}_n + H)$
- 5 : $\left[0, O^{max} + H \prod_{i=1}^{n} (C_i + 1)\right)$
- 6 : $\left[0, H \prod_{i=1}^{n} ((O_i + D_i - T_i)_0 + 1)\right)$

During the implementation of the tree, each edge will be a condition and a verification in order to guarantee the validity of the path and so, the interval chosen. The decision tree above shows only the path leading to the intervals which their algorithm will be added to the framework and nothing more. This means that the intervals paths that are not represented in the tree aren't pertinent in the way that they didn't bring a proof on the computed simulation.

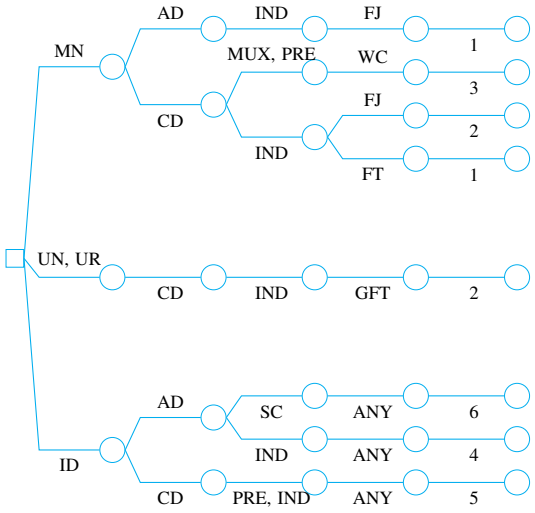

Fig. 2. Simulations intervals decision tree

V. Implementation and Evaluation

In this section we first explain the implementation of the simulation intervals algorithms. Second, we present its validation.

A. Implementation into Cheddar

As shown in the previous section, the decision tree was implemented into a specific *Ada* package in the Cheddar framework where all simulation intervals were implemented in dedicated functions. Each of these functions being called depending on the decision tree conditions. One of the issues encountered during the process of integration was the manipulation of big integers. A solution has been already tested in Cheddar for this issue but wasn't validated or even used for this type of calculation into the framework. This solution uses the package called *Big Numbers* and allows calculations on big integers. This solution was used in one algorithm and tested but issues have been encountered with the others algorithms that need calculation on floating point values which leads to conversions problems that haven't been resolved for now due to lack of time. However, on the others algorithms, the use of big integers have been validated as we will see in the next subsection. Another issue was the verification of the characteristics of the systems like the preemptivity, the processor types etc. and how to control them in order to call the corresponding interval algorithm. Ultimately, some of them were already present in the framework some others don't and have consequently been created. From a system and his formalization in DSL or *CheddarADL* the specifics intervals are called as expected.

B. Evaluation and validation

We have produced several experiments in order to evaluate the correctness of the calculations provided by the implemented algorithms.

To perform such evaluations, we have beforehand created multiple tasks set with various characteristics to target the multiple simulations intervals. Every calculation was verified multiple times and then, after the implementation of the solution in Cheddar, we have compared the results provided between the bare hand calculations and the ones provided by the algorithms in Cheddar. No error has been detected in the example provided to the program. Yet, as said in the previous subsection, even if a part of the algorithms has been validated with the use of big integers, some of them didn't. A short period of time would suffice to answer this issue.

VI. Conclusion

In this article, we introduced a way to improve the simulations interval boundaries for different tasks systems based on multiples works in real-time systems research papers. This new adaptive feasibility interval allows us to check the schedulability of a system. Those intervals allow multiples types of systems covering characteristics as deadlines types, dependencies, scheduling algorithms types used or processors

types. All these elements are taken into account to ensure a provable model of the system.

After research on the subject and the creation of several tasks set for tests and validation, a conceptual analysis of the solution has been made in order to have a later proper implementation.

The various intervals and their algorithms have been implemented into Cheddar. To ensure the security of the implemented code multiple functions and code structures proper to Ada have been used to prevent conversions, size or type issues. From a set of tasks with its characteristics, it is then possible to get the scheduling interval ensuring - if no issue is encountered within it - a schedulable system. The algorithms implemented for intervals calculations have been tested and validated on multiple tasks set examples.

This new functionality allows the Cheddar's user to have a boundary on the simulation of his system that guarantees the schedulability within the calculated interval.

In future work, this functionality should be integrated into the Cheddar GUI while it's currently implemented in the Cheddar kernel.

ACKNOWLEDGMENT

I would like to thank Frank Singhoff from UBO/Lab-STICC and his help on the Cheddar kernel. This work is done for the end-of-studies project.

REFERENCES

[1] F. Singhoff, J. Legrand, L. Nana and L. Marcé. "Cheddar: a flexible real-time scheduling framework." *ACM SIGAda Ada Letters*, vol. 24, no. 4, pp. 1-8, Dec 2004

[2] J. Goossens, E. Grolleau, L. Cucu-Grosjean, "Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms", *Real-Time Sys,* 52:808-832, 2016

[3] J. Goossens, R. Devillers. "Feasibility intervals for the deadline driven scheduler with arbitrary deadlines." *Proceedings of the 6th IEEE international conference on real-time computing systems and applications*, pp 54-61, 1999

[4] Big Number Ada Package, URL:http://bignumber.chez.com/info.htm

[5] Y. Ouhammou , E. Grolleau, M. Richard, P. Richard, F. Madiot, "MoSaRT Framework: A Collaborative Tool for Modeling and Analyzing Embedded Real-Time Systems", *Complex Systems Design & Management*, pp 283-295, 2015

[6] C. Fotsing, F. Singhoff, A. Plantec, V. Gaudel, S. Rubini, S. lI? h; n. Tran, L. Lemarchand, P. Dissaux, and J. Legrand, "Cheddar architecture description language," *Lab-STICC Technical report,* 2014

[7] J. Xu, DL. Parnas, "Priority scheduling versus pre-run-time scheduling," *Real Time Syst,* 18(1):7-23, 2000